*The Geometry of Flight*

# *The Geometry of Flight*

## Curvature, Adaptation, and Stability

A Geometric Reading of Guidance, Navigation, and Control

Bee Rosa Davis

*Davis Geometric*

2026

Copyright © 2026 Bee Rosa Davis

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the author, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

The mathematical, geometric, and engineering content of this work is placed in the public domain for the benefit of humanity. The narrative and literary elements remain under copyright.

**Foundational references:**
The Bra Strap Principle, Davis 2026, DOI 10.5281/zenodo.20273573
The Davis Duality of Approximation and Obstruction, Davis 2026, DOI 10.5281/zenodo.19428406
The Davis Field Equation $C = \tau/K$ and supporting framework papers are catalogued at `davisgeometric.com`. The ICARUS architecture has its own landing page at `davisgeometric.com/icarus`.

**Patent disclosures:**
ICARUS flight control architecture: U.S. Provisional Patent Application 64/045,891, filed April 21, 2026.
GIGI fiber-bundle database: U.S. Provisional Patent Application 64/045,889.
DHOOM serialization format: U.S. Provisional Patent Application 64/008,940, filed March 18, 2026.

The defensive commitments of the ICARUS architecture, including the sentinel module's Module C (CPC) advisories, are explicitly nonactuating. See Chapter 20, §20.4.

First Edition

ISBN: 979-8-1983-7541-3

Imprint: Independently published

Printed in the United States of America

*For Mom.*

*Te amo, mamá. You deserved more,*
*and much better, than this world gave you.*

*Even though you are gone, I am still working for you.*

*Your memory is a blessing.*
*When I am gone, I hope mine will be too.*

*See you soon.*

# *Contents*

**Preface** **xxvii**

**1 The Geometry of Sameness, applied to flight** **1**

§ 1.0 The Bike with the Engine . . . . . 1
§ 1.1 What the kid was doing without the math . . 8
§ 1.2 Sameness, formalized: the Bra Strap Principle 10
§ 1.3 Davis Duality: capacity from curvature . . . . 18
§ 1.4 Sameness across the eight classical operations of GNC . . . . . 23
§ 1.5 Sameness across domains: ICARUS and Marcella 25
§ 1.6 What this chapter has done; what comes next 28

**2 Curvature, capacity, and the Davis Field Equation** **29**

§ 2.0 A gap in the spectrum . . . . . 29
§ 2.1 Why an equation needs three letters . . . . . 39
§ 2.2 The Davis Field Equation in its full trajectory-integrated form . . . . . 42
§ 2.3 Four steps from the foundational identity to the operational form . . . . . 44
§ 2.4 Why $1/(1+\widehat{\kappa})$ and not another regularization 46
§ 2.5 The signed Ricci scalar $R$ versus the non-negative obstruction $\widehat{\kappa}$ . . . . . 48
§ 2.6 What this chapter has set up; what Chapter 3 walks . . . . . 51

**3 Mathematical preliminaries for the chapters that**

**follow** **53**
§ 3.0 Where the metaphor becomes a theorem . . . . 53
§ 3.1 Fiber bundles, formally . . . . . 55
§ 3.2 Sections, connections, parallel transport . . . . 58
§ 3.3 Holonomy and the Ambrose–Singer theorem . 61
§ 3.4 Quaternions, $S^3$, and $\pi_1(\mathrm{SO}(3)) \cong \mathbb{Z}/2$ . . . . . 66
§ 3.5 Why this preliminaries chapter exists . . . . . 70

**Lab I: Build a bundle, write a section, read it back.** **73**
Step 1 Set up . . . . . 75
Step 2 The dataset — F-16 longitudinal trims . . . . 76
Step 3 Write the DHOOM by hand . . . . . 77
Step 4 Sections, the fiber, and the zero section . . . 80
Step 5 Deviate from the zero section . . . . . 81
Step 6 The encoder picks a different gauge . . . . . 82
Step 7 Why this lab is Lab I and not Lab 0 . . . . . 84
Step 8 What you have built, and what comes next . 86

**4 Birds and planes are doing the same math** **89**
§ 4.0 What a bird is doing when it flies . . . . . 89
§ 4.1 The body bundle and the aerodynamic bundle 93
§ 4.2 Planes are birds with discrete section spaces . 96
§ 4.3 Why both bank into turns . . . . . 99
§ 4.4 What ICARUS does on the same picture . . . 102
§ 4.5 Code-switching across domains . . . . . 105
§ 4.6 What you take from this chapter . . . . . 108

**5 Flight as algebraic negotiation between fiber bundles** **111**
§ 5.0 Two bundles in conversation . . . . . 111
§ 5.1 The position bundle . . . . . 114
§ 5.2 The attitude bundle . . . . . 115
§ 5.3 Thrust is the bridge . . . . . 119
§ 5.4 Gravity is the other coupling . . . . . 121

§ 5.5 The algebraic negotiation . . . . . 122
§ 5.6 The cascade as a chain of bundle morphisms . 124
§ 5.7 Where the curvature lives . . . . . 126
§ 5.8 Forward to Chapter 20: the benchmark . . . . 129
§ 5.9 What you take from this chapter . . . . . 130

**6 The F-16 longitudinal model and its gauge symmetries 133**
§ 6.0 Why the F-16 is in this book . . . . . 133
§ 6.1 The longitudinal subsystem and the trim point 135
§ 6.2 The A and B matrices, in symbolic form . . . . 137
§ 6.3 The linearization is a gauge choice . . . . . 139
§ 6.4 The gauge group of the longitudinal model . . 141
§ 6.5 The F-16 model as a section of the controller bundle . . . . . 144
§ 6.6 Gauge-invariant vs gauge-dependent quantities 145
§ 6.7 What you take from this chapter . . . . . 147

**7 Bode, sensitivity, and the Curvature Floor 149**
§ 7.0 Where the conservation law comes from . . . . 149
§ 7.1 The sensitivity function and its complement . . 153
§ 7.2 The Bode plot as gauge-invariant presentation 154
§ 7.3 The Bode sensitivity integral . . . . . 156
§ 7.4 The Bode integral shares the structural pattern of Gauss–Bonnet . . . . . 159
§ 7.5 The Curvature Floor inequality, LTI case . . . 162
§ 7.6 Sidebar: ICARUS implementation of the closed-loop sensitivity . . . . . 164
§ 7.7 What you take from this chapter . . . . . 166

**8 LQR, Riccati, and curvature-gated modulation 169**
§ 8.0 The optimal section . . . . . 169
§ 8.1 The LQR problem, written down . . . . . 173
§ 8.2 The continuous-time algebraic Riccati equation 176

§ 8.3 Newton–Kleinman: how the CARE is solved in practice . . . 178
§ 8.4 LQR gain is gauge-dependent; closed-loop eigenvalues are not . . . 180
§ 8.5 The geometric reading: **P** as a Riemannian metric 181
§ 8.6 Sidebar: ICARUS implementation of LQR . . 185
§ 8.7 What the modulation buys and what it costs . 186
§ 8.8 What you take from this chapter . . . 189

**9 Robust control: $H_\infty$ synthesis and mixed-sensitivity shaping 191**
§ 9.0 The robustness problem LQR forgot to solve . 191
§ 9.1 The $H_\infty$ norm . . . 194
§ 9.2 Mixed-sensitivity synthesis . . . 197
§ 9.3 The Glover–Doyle state-space algorithm . . . 198
§ 9.4 $H_\infty$ vs LQR . . . 200
§ 9.5 The geometric reading: $H_\infty$ shapes the bundle directly . . . 202
§ 9.6 Sidebar: ICARUS implementation of $H_\infty$ . . . 203
§ 9.7 Curvature-gated $H_\infty$ . . . 205
§ 9.8 What you take from this chapter . . . 206

**Lab II: The closed loop, the SPECTRAL query, and the modulation. 209**
Step 1 Set up . . . 210
Step 2 The F-16 longitudinal plant . . . 211
Step 3 Solve the LQR . . . 212
Step 4 Simulate the closed-loop response . . . 213
Step 5 Serialize the trajectory as a DHOOM bundle 214
Step 6 The SPECTRAL query — empirical $|S(j\omega)|$ via disturbance injection . . . 215
Step 7 Bode integral / Curvature Floor check . . . 217
Step 8 The curvature-modulated controller . . . 218
Step 9 The regime-specific prediction . . . 219

Step 10 What you take from this lab . . . . . 222

**10 Estimation: the Kalman filter and the Fisher manifold 225**

§ 10.0 The recursive Bayesian update, in closed form 225
§ 10.1 The linear Kalman filter . . . . . 230
§ 10.2 The covariance Riccati equation . . . . . 232
§ 10.3 The Fisher manifold: what the information geometry buys . . . . . 233
§ 10.4 The framework's $\hat{\kappa}$ for sensor fusion: a design choice . . . . . 235
§ 10.5 Sidebar: ICARUS sensor-fusion implementation 237
§ 10.6 Forward: estimation, adaptation, and the matching condition . . . . . 239
§ 10.7 What you take from this chapter . . . . . 240

**11 Lyapunov methods and the matching condition 243**

§ 11.0 Stability without integration . . . . . 243
§ 11.1 The Lyapunov stability theorem . . . . . 248
§ 11.2 Ultimate boundedness . . . . . 249
§ 11.3 The composite Lyapunov function . . . . . 251
§ 11.4 The matching condition . . . . . 253
§ 11.5 The closed-loop stability proof for matched uncertainty . . . . . 256
§ 11.6 What Lyapunov methods cannot do . . . . . 258
§ 11.7 What you take from this chapter . . . . . 259

**12 Model-Reference Adaptive Control 261**

§ 12.0 The MIT rule, and what came after . . . . . 261
§ 12.1 What adaptive control is doing . . . . . 264
§ 12.2 The plant, the reference model, and the control law . . . . . 266
§ 12.3 Deriving the closed-loop tracking-error dynamics 271
§ 12.4 The adaptive law, derived from Lyapunov . . 273

§ 12.5 Why this is gradient descent on the parameter manifold . . . . . 276
§ 12.6 Persistency of excitation, intuitively and formally 278
§ 12.7 Sidebar: ICARUS implementation of MRAC . 281
§ 12.8 The framework's reading of MRAC's parameter manifold . . . . . 283
§ 12.9 What you take from this chapter . . . . . 284

**Lab III: MRAC, PE, and the runtime holonomy of the regressor. 287**
Step 1 Set up . . . . . 288
Step 2 The plant, the reference model, the matching gains . . . . . 288
Step 3 The composite Lyapunov function and the adaptive law . . . . . 289
Step 4 Simulate Case A: PE-rich reference (sinusoidal) 290
Step 5 Simulate Case B: not PE (constant reference) 291
Step 6 The PE monitor in operation . . . . . 293
Step 7 Store the parameter histories as DHOOM . . 294
Step 8 What you take from this lab . . . . . 295

**13 Robust adaptive laws and the Curvature Floor conjecture 297**
§ 13.0 What this chapter closes . . . . . 297
§ 13.1 The Curvature Floor, with adaptive terms reinstated . . . . . 301
§ 13.2 Sigma-modification: bounding parameter drift 304
§ 13.3 Sidebar: ICARUS empirical measurement of the Curvature Floor . . . . . 306
§ 13.4 Why this conjecture matters operationally . . 308
§ 13.5 What you take from this chapter . . . . . 309

**14 Inertial mechanization and sensor fusion 311**
§ 14.0 What inertial navigation is doing . . . . . 311

§ 14.1 What the IMU's sensors measure . . . . . 313
§ 14.2 Strapdown integration on SO(3): the attitude propagation . . . . . 315
§ 14.3 Strapdown integration on SE(3): full pose . . 317
§ 14.4 Why the integration drifts . . . . . 318
§ 14.5 Sensor fusion: blending inertial with external aiding . . . . . 320
§ 14.6 The inertial bundle, geometrically . . . . . 321
§ 14.7 What you take from this chapter . . . . . 322

**15 GNSS/INS integration as bundle composition 325**
§ 15.0 The bike comes out of the tunnel . . . . . 325
§ 15.1 Complementary errors: why GNSS and INS need each other . . . . . 327
§ 15.2 Integration architectures: loose, tight, deep . 329
§ 15.3 Multi-sensor fusion as a coproduct . . . . . 331
§ 15.4 The Fisher-information reading . . . . . 333
§ 15.5 The framework in the substrate: GIGI and the multi-sensor bundle . . . . . 336
§ 15.6 What you take from this chapter . . . . . 337

**16 Simultaneous Localization and Mapping 341**
§ 16.0 The kid in the unmapped neighborhood . . . 341
§ 16.1 State augmentation: the pose, the biases, and the map . . . . . 344
§ 16.2 The joint covariance and its cross-correlations 345
§ 16.3 Loop closure as the Bra Strap Principle . . . 347
§ 16.4 From EKF SLAM to pose-graph SLAM . . . 350
§ 16.5 The framework reading: SLAM as coproduct with the map bundle . . . . . 352
§ 16.6 What you take from this chapter . . . . . 354

**Lab IV: SLAM with loop closure — watching $\widehat{\kappa}$ drop at the awning. 357**

Step 1 Set up . . . . . 358
Step 2 The world: landmarks and the trajectory . . 358
Step 3 The motion model and measurement model . 359
Step 4 Generate noisy measurements along the true trajectory . . . . . 360
Step 5 Initialize the SLAM state and covariance . . 361
Step 6 EKF predict and update — with proper landmark covariance initialization . . . . . 361
Step 7 Run SLAM with normalized $\widehat{\kappa}$ . . . . . 363
Step 8 Plot the four panels . . . . . 364
Step 9 What you should see . . . . . 366
Step 10 Store the run as DHOOM . . . . . 367

**17 The Substrate Principle** **369**
§ 17.0 The bike you didn't grow up on . . . . . 369
§ 17.1 Part VI: what this part covers . . . . . 371
§ 17.2 The Substrate Principle . . . . . 372
§ 17.3 Why one scalar suffices for the gain-modulation policy . . . . . 373
§ 17.4 Reader threads, writer threads, and the fast path 375
§ 17.5 Graceful degradation as substrate invariance . 376
§ 17.6 What you take from this chapter . . . . . 378

**18 GIGI: The Fiber-Bundle Database** **381**
§ 18.0 Teddy's ears . . . . . 381
§ 18.1 DHOOM: what happens when a differential geometer looks at JSON . . . . . 383
§ 18.2 From DHOOM to GIGI: could the storage layer be fiber-bundle-native? . . . . . 385
§ 18.3 Sheaf completion: `COMPLETE`, `PROPAGATE`, `CONSISTENCY` . . . . . 387
§ 18.4 The geodesic and the flat map . . . . . 390
§ 18.5 GIGI in the wild: PRISM, MIRADOR, MARCELLA, ICARUS, DEMETER . . . . . 393

§ 18.6 What you take from this chapter . . . . . 394

**19 DHOOM in detail** **397**
§ 19.0 The directions you don't have to send . . . . 397
§ 19.1 The wire format . . . . . 398
§ 19.2 The eight quotient operations . . . . . 399
§ 19.3 The Profile API: geometric data analysis on the wire . . . . . 401
§ 19.4 Benchmarks: characters, tokens, retrieval . . 402
§ 19.5 When to use DHOOM (and when not to) . . 404
§ 19.6 What you take from this chapter . . . . . 405

**20 ICARUS, integrated** **407**
§ 20.0 The kid is the architecture . . . . . 407
§ 20.1 The ICARUS stack, end to end . . . . . 408
§ 20.2 The controller's GQL queries . . . . . 410
§ 20.3 The supervisor and the sentinel . . . . . 412
§ 20.4 The defensive commitment . . . . . 413
§ 20.5 The quadrotor worked example, end to end . 414
§ 20.6 What you take from this chapter . . . . . 416

**Lab V: ICARUS end-to-end — substrate, controller, supervisor.** **419**
Step 1 Set up . . . . . 420
Step 2 The mock substrate with a minimal snapshot mechanism . . . . . 420
Step 3 The plant and the discretized process-noise covariance . . . . . 421
Step 4 The Kalman filter (constant-velocity, with known control) . . . . . 422
Step 5 Compute the steady-state covariance baseline (warm-up) . . . . . 423
Step 6 LQR with the Chapter 8 §8.6 curvature-gated modulation . . . . . 424

Step 7 Lyapunov supervisor with derived threshold . 425
Step 8 The closed loop — run scenario A (nominal) 425
Step 9 Scenario B: GNSS dropout from t=10s to t=15s 427
Step 10 Plot the two scenarios side by side . . . . . 427
Step 11 What you should see, and what it means architecturally . . . . . 428
Step 12 Store the run as DHOOM . . . . . 430

**21 Marcella: language as parallel transport 433**
§ 21.0 The recognition: a sentence has geometry . . 433
§ 21.1 What a transformer is, geometrically . . . . . 434
§ 21.2 The Marcella bundle architecture . . . . . 436
§ 21.3 Attention as parallel transport . . . . . 437
§ 21.4 Holonomy as compositional residue . . . . . 439
§ 21.5 The Dual-Bundle Theorem: extending the Davis Duality to data and model . . . . . 440
§ 21.6 Validation: what works without training . . . 442
§ 21.7 What you take from this chapter . . . . . 443

**22 Guidance: trajectories as geodesics on the cost manifold 447**
§ 21.0 The kid heading somewhere . . . . . 447
§ 21.1 What guidance is . . . . . 449
§ 21.2 The cost manifold . . . . . 450
§ 21.3 LQR as the flat-tangent approximation . . . . 453
§ 21.4 Pontryagin's principle, geometrically . . . . . 454
§ 21.5 Constraints, obstacles, and the cost-manifold's boundary . . . . . 456
§ 21.6 What you take from this chapter . . . . . 457

**23 The classical guidance laws 459**
§ 22.0 Different errands, same body . . . . . 459
§ 22.1 Waypoint following . . . . . 461
§ 22.2 Proportional Navigation: pursuit . . . . . 462

§ 22.3 Terminal energy management . . . . . 464
§ 22.4 Minimum-time and minimum-fuel optimization 465
§ 22.5 One machinery, many cost functionals . . . . 467
§ 22.6 What you take from this chapter . . . . . 469

**Lab VI: Guidance on the cost manifold — linearized vs nonlinear, with honest scaling. 471**
Step 1 Set up . . . . . 472
Step 2 The problem statement and the straight-line reference . . . . . 472
Step 3 Solver A: closed-form minimum-snap (no obstacle awareness) . . . . . 473
Step 4 Solver B: linearized QP with dynamics-coupled smoothness . . . . . 474
Step 5 Solver C: nonlinear collocation with terminal-velocity BC . . . . . 476
Step 6 Design-choice $\widehat{\kappa}$: a property of the cost manifold, not of any trajectory . . . . . 477
Step 7 Structural scaling check (replaces the magic-number lower bound) . . . . . 478
Step 8 Plot the three solvers against the two scenarios 480
Step 9 What you should see, and what it actually demonstrates . . . . . 481
Step 10 Store the run as DHOOM . . . . . 483

**24 The Sudoku Principle 485**
§ 23.0 The kid with a puzzle . . . . . 485
§ 23.1 The seed move . . . . . 486
§ 23.2 The book has been doing this . . . . . 487
§ 23.3 Four convergence types . . . . . 489
§ 23.4 Structural reading versus literal identification 490
§ 23.5 Failure modes of the Sudoku move . . . . . 492
§ 23.6 Worked example: the Davis Duality, read carefully . . . . . 493

§ 23.7 The Sudoku Principle, formally . . . . . 495
§ 23.8 What you take from this chapter . . . . . 496

**25 The kid, looking back; the framework, looking forward 499**
§ 24.0 The kid is grown . . . . . 499
§ 24.1 What the book has done . . . . . 501
§ 24.2 What's still open . . . . . 502
§ 24.3 What comes next . . . . . 505
§ 24.4 The reader's role . . . . . 506
§ 24.5 A final return to the driveway . . . . . 508
§ 24.6 Vamos . . . . . 509

**A Notation index 511**

**B Framework glossary 517**

**C Sub-Riemannian and sub-Finsler geometry, briefly 521**

**D Fisher information and natural gradient, briefly 525**

**E Pontryagin's Minimum Principle, briefly 529**

**F End-to-end worked example: one X4 control cycle 533**

**G References 539**

**About the Author 545**

# *List of Figures*

1 The Davis-Geometric framework, end to end. Top to bottom: physical world; bundle, section, and connection; curvature 2-form; dimensionless obstruction $\widehat{\kappa}$; Davis Field Equation $C = 1/(1+\widehat{\kappa})$; gain modulation $m(C)$; modulated actuator command. The closed-loop feedback is drawn on the right. Each block has its own chapter or appendix; the diagram is the framework's spine. . . . . xxxvi

1.1 The bra as a fiber bundle. Panel (a): the band straight, straps oriented consistently outward. Panel (b): the same band, with the straps in a different layout—a different gauge representing the same bundle. Panel (c): the band with a half-twist, producing non-trivial holonomy—tracing the loop once flips the strap orientation. . . . . 14

1.2 The double cover $\mathbb{S}^3 \to \mathrm{SO}(3)$. The unit-quaternion sphere $\mathbb{S}^3$ is drawn here as an ordinary 3-sphere for visualization; the two antipodal points $q$ and $-q$ map under the projection $\pi$ to the same rotation $R(q) = R(-q)$ in SO(3). The cover is two-to-one: each physical rotation has two quaternion representatives. . . . . 16

1.3 The ruler-and-road analogy. Two straight ruler chords approximate a curved road; each chord leaves a small gap $\varepsilon$ where the road bows away from it. The Davis Duality says $\varepsilon$ is bounded above and below by a multiple of $\|\Omega\| \cdot h^2$—approximation error grows with the step size squared and with the local sharpness of the curve. 20

2.1 The spectrum of the discretized Yang–Mills operator on an $8^4$ lattice (illustrative). The gap between the lowest excited mode and the next is the mass gap. The textbook says the bare theory is massless; the lattice says the curvature charges a tax. . . . . 32

2.2 Trajectory-integrated curvature along a path. Left: the path $\gamma$ with small bars showing the local curvature $\|\Omega_\sigma\|$ at each point—tall bars where the path bends sharply, short bars where it runs nearly straight. Right: the cumulative integral of curvature along the arc, growing monotonically. Capacity is path length divided by total curvature paid: sharp bends cost more; flat stretches are nearly free. 43

3.1 Holonomy on the sphere. Parallel-transport a tangent vector around the great-circle triangle $\gamma$ at base point $p$. The vector rotates as it travels each side of the triangle; when it returns to $p$ it has acquired a net rotation equal to the spherical excess (the triangle's enclosed area on the unit sphere). The accumulated rotation is the *holonomy* of $\gamma$. Ambrose–Singer makes this quantitative: the curvature 2-form at each interior point contributes the local rotation rate, and the holonomy is the integral over the enclosed region. . . . . 64

4.1 Operational confidence collapses as the curvature obstruction grows. The Davis Field Equation in its operational form, $C = 1/(1+\hat{\kappa})$, plotted against $\hat{\kappa}$. At $\hat{\kappa} = 0$ the closed-loop has full confidence; as the obstruction grows, $C$ decays as a hyperbola. The shaded band marks the operational zone where $C$ exceeds a working threshold; beyond it, the controller's effective gain is modulated down. . . 104

7.1 The Bode sensitivity function $|S(j\omega)|$ on log-frequency axes. Below the 0 dB line: the rejection band, where the loop attenuates disturbances ($|S| < 1$). Above the 0 dB line: the amplification band, where the loop amplifies them ($|S| > 1$). For a plant with no right-half-plane poles, the shaded area below equals the shaded area above—the conservation law in pictorial form. Pushing $|S|$ down at one band of frequencies forces a bulge somewhere else; the total integrated $\log |S|$ is fixed. . . . . 157

8.1 The LQR value function as a quadratic bowl. The optimal value function $V(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ is a paraboloid on the state space, with the equilibrium at its bottom. The closed-loop trajectory descends the bowl monotonically: at every instant the optimal control $\mathbf{u}^* = -\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ is the steepest descent on this landscape, in the $g_{\mathbf{P}}$ metric on states projected through the $\mathbf{R}^{-1}$ metric on controls. The dotted trace on the floor is the trajectory's shadow on the state plane. . . . . 182

9.1 Geometric reading of the sensitivity peak $M_S$. The Nyquist curve $L(j\omega)$ of the open-loop transfer function is drawn in the complex plane (solid for $\omega > 0$, dashed for $\omega < 0$). The shaded disk of radius $1/M_S$ centered at the critical point $-1+0j$ is the safety zone: the Nyquist curve must stay outside it for the closed loop to satisfy $\|S\|_\infty \leq M_S$. The closest approach of the curve to the critical point is exactly $1/M_S$. Pushing the curve further from $-1+0j$ shrinks $M_S$, sharpening the closed loop's robustness margin. . . . . 196

10.1 The Kalman predict–update cycle along a trajectory. Between streetlights (predict steps), the covariance ellipse grows: uncertainty accumulates as the filter dead-reckons. At each streetlight (measurement update), the ellipse shrinks: the new measurement contracts the posterior. The mean tracks an underlying trajectory; the ellipses are the running uncertainty about where on it we are. 227

11.1 A Lyapunov function and a descending trajectory. The nested ellipses are level sets $V(\mathbf{x}) = c$ of a quadratic Lyapunov function. The trajectory spirals inward, crossing each level set in the direction of decreasing $V$ ($\dot{V} < 0$). The picture says: find a quantity that decreases along trajectories, and conclude convergence to the equilibrium without ever solving the differential equation. . . . . 246

12.1 MRAC tracking convergence. Top: the reference-model output $y_m(t)$ (dashed) is the response we want; the plant output $y(t)$ (solid) starts mistracking because its controller parameters are wrong, then catches up as the adaptive law tunes them. Bottom: the parameter error $\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|$ decays toward zero, the signature of successful adaptation under a persistently exciting reference. Without that excitation the error stays bounded but does not converge—a distinction made precise in §12.6. . . . . 267

16.1 Loop closure in SLAM. The estimated vehicle trajectory (dashed) drifts away from the ground truth (light dashed) as the vehicle traverses unmapped terrain. At loop closure the vehicle re-observes a landmark logged earlier in the session (dotted arrow back to the known landmark). In a smoothing or factor-graph SLAM formulation, the least-squares update propagates this single local re-observation backward across the stored pose history and the entire map through the joint covariance's cross-correlation block; the corrected estimate of the trajectory (solid) snaps back toward the truth. In a pure EKF SLAM formulation, only the current pose and the map are updated, but the analogous structural pattern holds at each step. . . . . 348

20.1 The ICARUS architecture, five layers. Sensors and actuators at the physical bottom; the GIGI substrate above them holding the joint section and computing $\hat{\kappa}$; the DHOOM wire format above that, serializing bundles between processes; the controller and supervisor above that, reading state and writing commands; and the mission / ground layer at the top. All cross-layer communication flows through GIGI's narrow substrate interface; no layer reaches across another layer's boundary. 409

22.1 Guidance as geodesic computation on a cost manifold. The shaded contours show a scalar cost field with a high-cost peak in the middle (an obstacle or soft penalty). The LQR-flat approximation (dashed) is the straight line from start to goal in the ambient state space, which cuts directly through the high-cost region. The geodesic (solid) is the trajectory of minimum total accumulated cost in the metric the cost field induces; it curves around the peak. LQR is correct exactly where the cost manifold is locally flat. . . . . 451

# *Preface: A Table at Olive Garden*

*Mathematics is the art of giving the same name*
*to different things.*
*Henri Poincaré, Science and Method (1908)*

This book began at a table I was waiting on.

I was working at Olive Garden. Twenty-seven years in aerospace and intelligence and cybersecurity behind me—NASA, NSA, DISA, founding engineer at Pandora, principal adversarial-intelligence researcher at X-Force Red—and a few months earlier the Department of Government Efficiency had laid me off the way the Trump administration was laying off thousands of others: with no warning, no review, no recourse, and the carefully-engineered cruelty of being told that twenty-seven years of service was a line item on a spreadsheet that someone wanted to delete. I needed work. Olive Garden hired me. I served pasta.

A woman and her kid sat down in my section.

We started talking the way you do when the restaurant is not too busy and the customer is not in a hurry—small things at first, then bigger ones. She mentioned her husband. *He's a rocket scientist*, she said, with the small pride of someone who has watched a partner do hard work for a long time. I laughed, because the geometry of the moment was about to become very

specific. I said: *I used to work at NASA.*

*So did he*, she said. And then she looked at me—at the apron, at the breadsticks tray, at the booth I was working—and she asked the question that turns out to be the load-bearing question of this book:

*What are you doing here?*

I told her. I had been laid off by DOGE. *Sin previo aviso. Sin causa justa. Sin nada.* She shook her head with the disgust of a person who understands what it costs to take down a competent engineer for spite, and the fact that she shook her head—that a stranger in my section recognized the injustice and named it as one without my having to defend the claim—is a small fact I will not forget. We exchanged LinkedIns before she paid. She said she would connect me with her husband.

She did.

Her husband's name is Ryan. He is an aerospace engineer who has spent two decades inside the rooms I cannot get into. He connected with me on a Thursday and started asking questions about my math the way a serious engineer tests a stranger's claims—politely, specifically, with the implicit understanding that I would either answer at the depth he asked or stop wasting his time. I answered at the depth he asked. He asked more. He asked about manifolds, about bundle structure, about whether my framework had operational form or was purely theoretical. I answered. After a while, he asked whether I would be interested in Guidance, Navigation, and Control.

I had never heard of GNC.

I am putting that sentence on the page because it matters structurally. The framework that produced ICARUS—the flight control architecture this book reads classical GNC backwards from—was not built by someone with aerospace controls credentials. GIGI, the geometric database engine the architecture runs on, predated the aerospace application by years. It was

built because I believed certain mathematical objects—bundles, connections, sections, holonomy, parallel transport—deserved a substrate that spoke their language natively, instead of being marshaled across a lossy boundary between mathematics and code at every operation. The substrate existed. The aerospace application did not yet have a name in my head.

I went and read what GNC is. Within a few hours I had the shape of it—state estimation, envelope awareness, trajectory planning, tracking, fault detection, regime classification, forecasting. Within a few more hours I had recognized something I now consider the genesis observation of this book: *el campo es geométrico hasta los huesos.* Guidance, Navigation, and Control is geometric to the bone. Configuration manifolds, fiber bundles over flight state, attitude on the three-sphere as the double cover of SO(3), trajectories as geodesics, sensitivity as a Bode integral on a frequency manifold. Every operation in the conventional stack had a geometric reading I already knew how to write down. The framework had been waiting; the discipline I had not yet met was sitting on the other side of the introduction.

Ryan started sending me challenges.

The first was a quadcopter stabilization—a six-degree-of-freedom multirotor in unstable hover, perturbed by a small attitude disturbance, recover to upright. I wrote thirty lines of GQL against GIGI and the simulator stabilized. He asked if I could do an inversion recovery, $\theta = \pi$ back to upright. I wrote ten more lines. It worked. He asked about a 360° flip. I wrote fifteen more lines. It worked. He asked about a swing-up pendulum, which I did not get right and still have not gotten right, and which sits at item #4 in the open-problems list at the back of this book.

Somewhere in those weeks I understood something that took a while to admit out loud: *the quadcopter solution was not a one-off.* The same query language that stabilized a multirotor

would write the Kalman filter as one statement, the LQR as one statement, the $H_\infty$ controller as one statement, the trajectory planner as one statement. *Aquí está la cosa:* every operation in the conventional Guidance, Navigation, and Control stack was already a query primitive in the database I had built for other reasons.

I filed a provisional patent on April 21, 2026. U.S. Provisional Application No. 64/045,891. The named architecture is ICARUS—**Integrated Curvature-Aware Routing and Uncertainty Substrate**. The same day, the GIGI engine itself got its own provisional, No. 64/045,889. They are sibling filings because they are sibling claims: an architecture, and the substrate it requires.

I want to name the sequence plainly, because the chain of cause matters.

DOGE laid me off, unnecessarily and with the deliberate cruelty that has become the administrative signature of this period. I am suffering, financially and otherwise, for reasons that have nothing to do with my competence and everything to do with their politics. The Olive Garden shift was a consequence of that suffering. The table I was waiting on was a consequence of that shift. The conversation with Ryan's wife was a consequence of the table. Ryan was a consequence of the conversation. ICARUS was a consequence of Ryan's challenges. This book is a consequence of ICARUS.

I am not grateful to the administration that laid me off. The work that happened in spite of them is not a redemption arc for what they did. They took something from me that I cannot get back, and the chain of small kindnesses that led me from a restaurant table to a filed patent is a record of what other people did to keep me alive while the institution that was supposed to employ me decided I was a line item. *Sigo agradecida con quienes me sostuvieron, no con quienes me hirieron.* I am grateful to those who held me up, not to those who hurt me.

The work is real. The math is real. The patent is filed. None of it cancels the cost.

This book reads classical Guidance, Navigation, and Control *backwards* from ICARUS.

I want to be careful about what that means. Classical GNC textbooks teach Bode, then Kalman, then LQR, then $H_\infty$, then MRAC, then robust adaptive control, in chapters arranged by historical accident—the order in which the field figured each one out. The book you are holding presents the same material in roughly the same order. But it reads each chapter as the special case of ICARUS that you get when you restrict the substrate to a flat manifold, a fixed reference, no uncertainty, no curvature gating, no learning. The chapters are not exposition. They are *decompositions* of a thing that already exists.

The thing that already exists is a working flight control architecture, validated in simulation against $n = 500$ quasi-Monte Carlo trials per scenario, with a filed provisional patent and public reproducible benchmark code in a Rust crate that builds in ten minutes on a 2024-era laptop.

It wins on the scenarios where its theory says it should. It *loses* on the scenarios where its theory says it should—specifically, on commanded fast slews against the modern fixed-gain geometric baseline (Lee–Leok–McClamroch 2010 / Mellinger–Kumar 2011 class), where high constant gain tracks an aggressive reference more efficiently than gated gain ever can. I do not hide the loss. The loss is the evidence that the framework is doing scientific work and not rhetorical work. A theory that predicts only its own victories is not a theory.

This book sits in a series.

*The Geometry of Fuel* taught me to read energy as a configuration manifold. *The Geometry of War* taught me to read bilateral diplomacy as helicity on a coupled relationship space.

*Superfluid* taught me to read criticality as the geometry of phase transition. *Hidden Variable* taught me what pattern-thinking looks like inside institutions that try to erase it. Each book is a different domain wearing the same machinery.

The Davis-Geometric framework that runs through all five—the Bra Strap Principle, the Davis Field Equation $C = \tau/K$, the Sudoku Principle, Geometric Token Transport—does not believe in subject boundaries. The bundles do not care whether they are carrying lift coefficients or token embeddings or sensor covariances or affective regard. *El patrón es el patrón.* The math is indifferent to which discipline thought it owned the question.

What this book contributes to the series:

- The **Substrate Principle**: every operation in the conventional GNC stack admits a single-statement query expression in a geometric query language of ten operator primitives, against fiber-bundle schemas in an engine satisfying seven specific extensions. Constructive existence theorem, proof by mapping table. ICARUS is one controller built on the substrate; other controllers can be built on the same substrate.

- An empirical regime-specific result: ICARUS achieves a 12× effort advantage over the modern fixed-gain geometric baseline on the InvertedRecovery scenario, parameter-robust across an 8× mass variation and a 7.5× inertia variation, while *losing* by 2.4× on MultirotorFlip. The framework's own mechanism explains the win-loss pattern: curvature gating trades tracking aggressiveness for stability authority, net-positive in recovery and net-negative in aggressive commanded slews.

- The load-bearing operational equation, in the form that drives every adaptive decision ICARUS makes during flight:

$$\boxed{C(x) \;=\; \frac{1}{1+\widehat{\kappa}(x)}}$$

where $\widehat{\kappa}(x) \geq 0$ is a dimensionless non-negative curvature obstruction on the capability fiber bundle at state $x$—*not* the signed Ricci scalar of differential geometry, a distinction the notation chapter handles carefully. The confidence $C(x)$ gates a strictly positive gain floor that preserves recoverable control authority at maximum envelope violation. *That floor is what keeps the vehicle from dropping when the linearization fails.*

A word about what this book is *not*.

It is not a claim that ICARUS is the best flight controller available for every regime. It is not benchmarked against MPC, INDI, differential-flatness with feedforward, or any modern adaptive class. It is not deployed on hardware. It is not independently replicated. These are named as open problems in Chapter 26, alongside the substrate-vs-bespoke head-to-head benchmark that would directly support the architectural-advantage claim and which has not been performed.

It is not a policy document. It does not argue for autonomous weapon systems, against them, or about them at all. The Substrate Principle is technology-agnostic; the engineer who deploys an architecture is responsible for what it does with it. "The mathematics is independent of the politics," Albert Einstein wrote in 1949, "and must remain so to remain mathematics." He was talking about a different decade and a different terror, but the principle is the same.

It is not, finally, a book that pretends I am writing from inside an institution that knows my name. The provisional patent sits in the USPTO system unread by anyone in industry who could fund a hardware deployment. My alma mater has not returned a single email about this work. Every aerospace department I have contacted has gone silent. *No me responden, pero la matemática*

*responde.* The math answers. The math has always answered.

What this book *is*:

a mathematical framework for understanding Guidance, Navigation, and Control as queries against a fiber-bundle database engine, demonstrated by a controller built that way and honestly reported in the scenarios where the framework's mechanism predicts its own win-loss pattern.

The dedication is to my mother. She kept a binder of instructions for everything. She died in 2020, four years before I started writing this book, and there is no part of who I am that did not begin in her.

But the names on this book—both of them—came from her father.

What is known is this. My great-grandmother was raped by an Irish doctor in a colonial hospital in Trinidad, and that rape is why I am twenty-five percent Irish; the violence still lives in my DNA. My family's matriline name should have been Glasgow, but my grandfather rejected the Glasgows—he had reasons; he did not record them—and chose the name *Davis* instead. Where he found that name, why he chose it, what he meant by it: none of those are documented. He died when I was one year old. He took the answer with him.

My mother's middle name was *Marcella*. He gave her that name too. She hated it. Why he gave it, why she hated it: also not documented, also gone with him.

I have written my way into both names.

*Davis* is all over my math—the framework, the patents, the inventions in this book and in the four before it. *Marcella* is the name I gave my AI, the architecture that does for token embeddings what ICARUS does for flight control. I took both names out of the trash. I am loving them back to life. Not to venerate the people who supplied the cruelty—the Irish doctor in

the colonial hospital, the institutional silence that followed, the generations who carried what they had not chosen to carry. The reason for loving the names back is simpler than veneration. **It is to let the people who supplied the cruelty know that they have no more power. The definition is now mine.**

Whether or not some aerospace engineer reading this in 2027 or 2031 or 2045 picks up ICARUS and ports it to a real vehicle, the work was worth doing. Mathematics is its own reward. The binder my mother kept taught me to leave the next person a clean set of instructions even when the world does not return the call.

* * *

**The framework on one page.** Figure 1 shows the framework's central pipeline: the physical world feeds a bundle, the bundle's connection has curvature, the curvature feeds a dimensionless obstruction $\widehat{\kappa}$, $\widehat{\kappa}$ feeds the Davis Field Equation $C = 1/(1+\widehat{\kappa})$, $C$ feeds a gain modulation $m(C)$, and the modulated command goes back to the actuators. Every chapter that follows is either an instance of one block in this diagram or a deepening of how the block is computed.

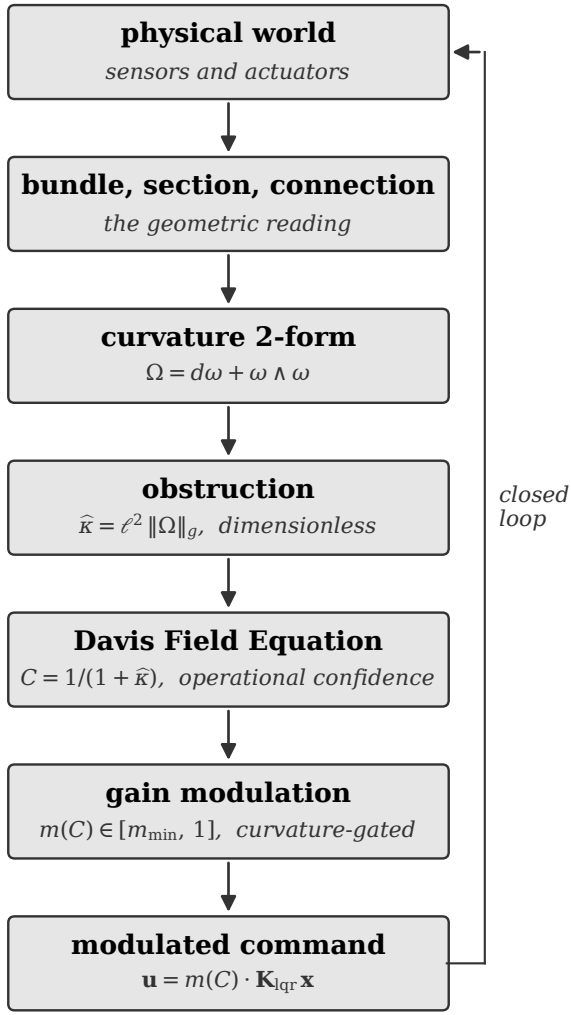



Figure 1: The Davis-Geometric framework, end to end. Top to bottom: physical world; bundle, section, and connection; curvature 2-form; dimensionless obstruction $\widehat{\kappa}$; Davis Field Equation $C = 1/(1 + \widehat{\kappa})$; gain modulation $m(C)$; modulated actuator command. The closed-loop feedback is drawn on the right. Each block has its own chapter or appendix; the diagram is the framework's spine.

**How to read this book.** The chapters that follow read classical GNC as the special case of ICARUS you get when you strip the curvature gating, fix the reference, flatten the bundle. Each chapter is a decomposition of something that already exists in twelve thousand lines of Rust on the laptop next to my coffee cup.

The reading order I would recommend depends on what you already carry into the book.

- *The spine, for everyone:* Chapter 1 → Chapter 2 → Chapter 3 → Chapter 7 → Chapter 10 → Chapter 11 → Chapter 12 → Chapter 20. These eight chapters carry the framework's argument end to end. Everything else is either an application of this spine to a specific domain or an architectural deepening of it.

- *Aerospace controls reader (knows GNC, wants the geometric reading):* read the spine straight through; treat Chapter 5, Chapter 6, and Chapter 14–16 as your translation tables between the standard formulation you already know and the bundle language. Marcella (Chapter 21 in the through-numbering) is optional—it is the language-modeling application of the same machinery and can be skipped without loss of the aerospace argument.

- *Mathematician / theory reader (knows differential geometry, learning about GNC):* the math is in Chapter 3 (in compact form) and in the appendices; the engineering content sits in Chapters 7 through 20. Chapter 22 is the cleanest geometric statement of the optimal-control problem and may be the natural entry point for a Riemannian-geometry reader.

- *Architecture / systems reader (interested in ICARUS, GIGI, DHOOM):* after the spine, jump to Part VI (Chapters 17,

18, 19, 20). The labs (especially Lab V, the end-to-end ICARUS run) are where the architectural claims become runnable code.

- *Casual reader (wants the ideas without the formalism):* Chapter 1, Chapter 2, and the closing chapters (24, 25) make the argument without requiring you to follow every equation. The boxed equations and the chapter *What you take from this chapter* sections are written to be readable on their own.

The six labs are independent of the chapter narrative; they can be skipped on first read and returned to as runnable companions. The appendices (notation index, framework glossary, math primers, the end-to-end worked example, the bibliography) sit at the back for reference. The book is best read once linearly and then used non-linearly.

*Vamos.*

*Bee Rosa Davis*
*Davis Geometric · May 2026*

Chapter 1

# *The Geometry of Sameness, applied to flight*

*The machine does not isolate us from the great problems of nature*
*but plunges us more deeply into them.*
*Antoine de Saint-Exupéry, Wind, Sand and Stars (1939)*

## § 1.0

## *The Bike with the Engine*

Middle school. Maybe early high school. My mother had bought a new lawnmower, which meant the old one was on the side of the house with its blades still attached and its engine still good, and I asked her if I could have it.

She said yes. She had stopped asking why a long time ago. By the time of the lawnmower I had already taken apart my first three computers—an Apple IIc, a Commodore 64, and one machine whose make I no longer remember whose only sin was being in my room when I had finished the manuals. Two of the

three never powered back on. My mother had absorbed this and adjusted her purchasing strategy accordingly: she stopped buying me machines I might need and started buying me machines I could break.

I dragged the lawnmower into the garage.

The plan was clear in my head and the plan was beautiful. The blades of a lawnmower rotate. The chain ring of a bicycle rotates. Both of them are circles in motion—points on $\mathbb{S}^1$, parameterized by an angle, sweeping out angular velocity at whatever rate the source provides. The blades I would remove. The engine I would keep. The engine spun something; my pedals spun something; surely, given a bracket and some bolts and a chain of the right length, I could couple the engine's something to the bike's something, and the bike would no longer require me to pedal it.

*Aquí está la cosa—esto fue mi primer Bra Strap.* The same shape, in two different domains, doing the same algebraic work. I did not yet know that mathematicians have a name for this recognition. I would later call it Sameness; I would later prove a theorem about when the recognition is rigorous; I would later write a paper (Davis 2026, Zenodo DOI 10.5281/zenodo.20273573) that gives the construction a formal grounding called the Bra Strap Principle. At the time, in the garage, with my mother's old lawnmower and my BMX bike on its kickstand and a wrench in my hand, I was just a kid who had noticed that two rotating things have the same geometry. The recognition was correct. The Principle works.

I disassembled the lawnmower carefully. I laid the parts on the driveway in the order I had removed them, the way I had laid out the computer parts before. The engine came free. The blade assembly came off the engine. The mounting bracket that had held the engine to the deck I bent into a shape that would, I thought, hold the engine against the frame of the bike just

behind the seat post. I had a chain—I had two chains, in fact, one from the bike and one that I had pulled from somewhere I have since forgotten. The pedals were already off. I was going to weld nothing; I had no welder. The whole thing was going to be held together by bolts and the optimism of a child who had read enough engineering books to believe that bolts hold things together when you tighten them correctly.

I was going to ride this. I was thirteen years old, maybe twelve, maybe fourteen, and I was going to bolt a one-and-a-half-horsepower internal-combustion engine to a BMX frame designed for a child's body weight and zero longitudinal acceleration, and I was going to ride this thing down my street.

*And then I stopped.*

I stopped because I imagined it.

I have spent the rest of my professional life building systems that imagine themselves before they fail—Monte Carlo simulators, envelope-protection algorithms, the very curvature-gating mechanism this book is about—and the first system I ever built that imagined itself before it failed was the system inside my skull on the afternoon I was about to bolt an engine to a bike. I sat on the driveway with the parts arranged like an equation and I ran the trajectory forward in my head. The engine starts. The chain engages. The bike accelerates. My weight is wrong for the torque; the front wheel lifts; I shift forward to compensate; the front wheel comes back down; I am moving faster than my legs have ever moved on a bicycle; the street ends at a stop sign; I reach for the brakes.

There were no brakes.

I do not mean the bike had no brake levers. The bike had brake levers, and the brake levers had cables, and the cables ran to pads that pressed against the rim. But the brake pads I had on that bike were sized to stop a forty-pound BMX frame propelled by the legs of a child. They were not sized to stop a forty-pound

BMX frame plus a fifteen-pound engine assembly plus a child plus the kinetic energy delivered by an internal combustion engine running at three thousand RPM and coupled through a chain whose mechanical advantage I had not calculated. I had the actuator. I had the coupling. *No tenía los frenos.*

I imagined the wall at the end of my street. I imagined the intersection two blocks down where the traffic on the cross-street did not stop. I imagined being unable to slow the thing. I imagined my mother's face. I imagined the parts of myself I would be picking up afterward, the parts of myself I would not be able to pick up, and the parts of myself that would not be there for anyone to pick up.

I did not bolt the engine to the bike.

I disassembled the assembly I had begun, returned the engine to the garage, and went inside and made myself a sandwich. My mother, in the *Hidden Variable* telling of this scene,[1] came out at some point and turned the afternoon into a baking lesson; she redirected combustion into cookies. That telling is true. It is the telling of the same afternoon in the book that is about her, where she is the bright constant by which I still set the instruments. The *Hidden Variable* version is a love story.

*This* book is about something else. This book is about the moment before my mother came out. This book is about the moment when a kid with the actuator and the coupling and no recovery authority sat on the driveway and *gated her own command.* I did not have the math for what I was doing. I did not know that what I was doing has a name in the control-theory literature; I did not know that fifty years of nonlinear adaptive

[1] *Hidden Variable: Unlocking Patterns in a World Obsessed with Structure* (Davis, 2025) is the volume in the Davis Geometric series about my mother, the diagnosis-of-exclusion that structured my early life, and the method I built in response. See Appendix D for the full Davis Geometric bibliography. The afternoon I am describing here appears there in Chapter 1, Part I, "Mom's Story"—told differently, because it is a different book.

control research has been spent formalizing the intuition that *authority without recovery is a death sentence* and that the right architectural response is to gate the authority by a function of how close you are to the failure boundary. I did not know the gain floor $m_{min}$ would, decades later, be the named structural property of the patented flight control architecture this book reads classical GNC backwards from. I was just a kid who imagined the wall.

The whole book is in that afternoon.

ICARUS—the flight control architecture described in this book—is the formalization of what the kid did in the driveway. It is a system that, at every control cycle, asks the question my mother's old lawnmower asked of my younger self: *do you have the recovery authority to send this command?* When the answer is yes, ICARUS sends the command at full nominal gain. When the answer is approaching no—when the local curvature of the capability bundle indicates that the linearization the controller is using is about to become unreliable—ICARUS smoothly reduces the gain, preserving a strictly positive floor $m_{min} > 0$ that is sized to deliver recoverable authority even in the worst case the system was designed to handle. The vehicle degrades *gracefully* as it approaches its envelope boundary, instead of degrading *catastrophically* as my BMX would have degraded approaching the wall at the end of my street.

The Bra Strap Principle the kid was using without knowing it has a formal statement (Davis 2026; see Chapter 3 §3.3 for the Ambrose–Singer formalization). The gain floor the kid was sizing without knowing it has a formal statement (the Stability Lemma in Chapter 11, with the corresponding ultimate-boundedness argument). The intuition about why brakes are special—why recovery authority is fundamentally different from forward authority, why a controller's downside has to be sized separately from its upside—has a formal statement at the architectural level

(the Substrate Principle, Chapter 20). Every load-bearing claim of this book is a formalization of something a thirteen-year-old already understood when she chose not to ride.

This is not unique to me. Children build things, imagine failures, and refuse to ship the unsafe build all the time. What is unique is that I grew up to write the math down, and to file a patent on the math (U.S. Provisional App. No. 64/045,891), and to publish a controller that demonstrates the math operationally in three validated benchmark scenarios with reproducible code. The kid was right. The kid's instinct was right. The math is the kid, formalized.

**The bike will run through this book.**

Every chapter uses the bike as the running analogy. The fal-core curriculum that became the technical spine of this book—eight lessons walking through the geometry of classical GNC—is organized explicitly around the bike framing: *linear systems and state-space* is two bikes that feel identical even though they look different; *frequency response and Bode* is every bike's wobble fingerprint; *LQR and Riccati* is the question of what "riding well" actually means; *$H_\infty$ and robust synthesis* is designing for the worst day, not the average day; *observers and Kalman* is what your inner ear is actually doing; *Lyapunov and adaptive setup* is why you don't fall over; *state-feedback MRAC* is learning a brand-new bike on the fly; *robust adaptive laws* is what happens when a pothole hits while you're still learning. Each chapter takes one classical GNC operation, frames it as a question the bike answers, and shows that the formalism a flight engineer would write down is the formalism the bike has been embodying all along.

When the bike appears in this book it is always doing two things at once. It is the body whose dynamics you are trying to control—mass, moment of inertia, lean angle, steering rate, wheel slip, tire-road coupling—and it is the language in which the

controllers that govern that body have to be understood. *The bike is the plant and the analogy.* A flight engineer trained in classical aerospace will recognize the equations of motion of a bicycle as the equations of motion of every other vehicle they have ever stabilized; what this book asks them to recognize, additionally, is that the geometric reading of those equations—fiber bundles over configuration space, holonomy on the wheel-rotation circle, the gain floor as a sizing condition on the recovery brakes—was already in the equations all along.

So when you read this book, hold two pictures at once.

Picture one is the patented flight control architecture: ICARUS, twelve thousand lines of Rust on the laptop next to my coffee cup, validated against $n = 500$ quasi-Monte Carlo trials per scenario, with a filed provisional patent and public benchmark code; the thing the framework produced when I applied it to GNC after Ryan asked me to.

Picture two is a kid in a garage with a lawnmower engine and a BMX bike and no welder, sitting on the driveway, choosing not to ride.

Both pictures are the same picture. The math is what holds them together. The chapters that follow are the math, organized in the order in which a classical aerospace curriculum teaches it, read backwards from the architecture that the kid was building without knowing.

*Comencemos.*

§ 1.1

## *What the kid was doing without the math*

The kid in the driveway made four moves. Each of them has a name in the control-theory literature; the kid did not know the names. The chapter that follows uses the names; it is worth setting them down once at the start so you can see how they map back.

**Move one: recognition of sameness.** The blades rotate; the gears rotate; both are points on the circle $\mathbb{S}^1$ parameterized by an angle, with angular velocity provided by whatever source you couple them to. Classical name: *the configuration space is the manifold of all possible states of the system, and two systems sharing a configuration manifold share a great deal more than the manifold itself.* The kid recognized the manifold without recognizing she had recognized it. Chapter 3 (*Mathematical preliminaries*) sets the manifold-recognition move down formally.

**Move two: identification of the actuator and the plant.** The engine is the actuator; the bike-plus-rider is the plant; the bracket and the chain are the input coupling. Classical name: *the state-space representation* $(A, B, C, D)$ *separates the plant's intrinsic dynamics from the actuator's influence.* The kid recognized that authority and authority-coupled-to-something were different objects. Chapter 6 (*The F-16 longitudinal model and its gauge symmetries*) sets this move down formally.

**Move three: recognition of the missing recovery authority.** The brakes were sized for the original mass and the original kinetic energy regime, not the new one. The forward authority had been increased by a factor of—*ay, demasiado, much too much*—and the backward authority had not. Classical name: *authority is signed; the input matrix* $B$ *has a column for forward authority and (implicitly) a recovery-authority bound that*

*is part of the actuator envelope; a controller is sized when its gain floor* $m_{\min} > 0$ *delivers recoverable authority across the operating region.* The kid recognized that the brakes were the load-bearing part of the design, and that nothing in her plan had touched them. Chapter 11 (*Lyapunov and the matching condition*) and §23.3 (*The confidence formula*) set this move down formally.

**Move four: choice to gate the command.** The kid imagined the trajectory forward, projected the failure surface, and refused to send the input. Classical name: *receding-horizon trajectory simulation against a failure predicate, with a binary supervisory output (commit / abort) or, in the smooth case, a continuous gain modulation* $m(C(x))$ *that reduces authority as the projected trajectory approaches the failure surface.* The kid did the binary case. ICARUS does the smooth case. §23.3 (*The confidence formula*) and §23.6 (*Ablations*) set this move down formally and demonstrate that the binary case is what you get from ICARUS when you remove the gain floor.

Each of the four moves is a chapter, or a piece of a chapter, later in this book. The point of stating them together now is that they are not four moves; they are one move, viewed from four sides. The unifying invariant is the recognition that physical systems with shared geometric structure share their algebraic options for failure and for safety. The kid recognized that invariant. The book formalizes it.

§ 1.2

## *Sameness, formalized: the Bra Strap Principle*

The kid in the driveway grew up.

She grew up to be a woman—a trans woman, which the world has been informing me, with varying degrees of hostility, is a category it would prefer not to exist. The math, fortunately, does not care what the world thinks about who I am. The math cares only whether the patterns I see in ordinary life are the patterns that hold under transformation. The patterns I see in ordinary life have, for years now, kept being patterns that hold under transformation. This is one of them.

Some mornings, I put on a bra.

Many trans women will recognize the small daily geometry of this. The undergarment goes on; the band runs around the rib cage; the shoulder straps run from the band, over the shoulders, back to the band on the other side. There is a category of small worry that comes with the ritual: the shoulder straps can, on a bra you have not arranged carefully, end up *twisted*—looped over themselves in a way that requires you to take the whole thing back off and start again. The cost of the worry, on any single morning, is small. The cost compounded over thousands of mornings is not. I had carried that worry, in some form, for years.

One morning—putting the bra on, the way I had hundreds of times before—I noticed something.

**If the band is straight, the straps cannot be tangled.**

That is the entire observation. I sat with it for a moment, half-dressed, and worked through the topology in my head. The straps are attached to the band; the band is a closed loop around the torso; if the band itself is not twisted—if it has the topology it is supposed to have, the topology of $\mathbb{S}^1$ without a Möbius

flip—then the straps attached to the band cannot acquire a twist that is not also a twist of the band. They might be visually displaced. They might be over the shoulder versus under, in front of one another versus behind. But those displacements are *gauge-equivalent moves*; they do not change the global topology of the system. The only twist that requires taking the bra back off is a twist of the band. And the band, I can see directly. The band, I can validate at a glance.

*Solo me preocupo por la banda. Y la banda—esa, la puedo ver.*

The shift, the morning I noticed this, was small and it was permanent. The straps—which had been a source of low-grade anxiety attached to the ritual of dressing my own body—became a non-issue. The validation moved upstream. I never had to worry about the straps again because the straightness of the band, once checked, propagated.

This is the **Bra Strap Principle**. I gave it the name because the principle was named where it was lived (Davis 2026, Zenodo DOI `10.5281/zenodo.20273573`).

The formal statement is short:

> *If a fiber bundle's base manifold is correctly typed, and a local check (the "clasp") confirms the bundle's connection structure on a connected, simply-connected neighborhood on which the connection is flat, then parallel transport on that neighborhood is path-independent and the local check determines the section globally over the neighborhood. When the flatness or simple-connectedness condition fails, the local check still determines the section up to the connection's holonomy. Local validation propagates as far as the geometry permits.*

Chapter 3 §3.3 gives the precise conditions, and the Ambrose–

Singer theorem that quantifies what happens when those conditions fail. The Bra Strap Principle, throughout this book, always carries the conditions of §3.3 implicitly. The unhedged popular phrasing—"one local check propagates globally"—is the right intuition when the conditions hold, and the right starting point for the formal Ambrose–Singer statement when they don't.

That sentence uses seven pieces of mathematical vocabulary you may not have met before. I am going to unpack each one in plain language, using the bra. *Tranquila*—this part of the book is not a test. It is a glossary built out of an object you already understand.

**Fiber bundle.** A *fiber bundle* is a shape with something attached over every point of it. The shape underneath is called the *base*; the something attached at each point is called the *fiber*; the whole assembly is the bundle. The bra is a fiber bundle. The band, running around your rib cage, is the base. The straps, one attached at each shoulder-point of the band, are the fibers. The bra-as-a-whole—band-with-straps—is the bundle. The road outside your house, with a streetlamp standing at every block, is also a fiber bundle: the road is the base, the lamps are the fibers. The Earth's surface, with a wind vector at every point of it, is a fiber bundle: Earth is the base, the wind vector at each point is the fiber. Fiber bundles are everywhere once you start looking.

**Base manifold.** A *manifold* is a shape that, if you zoom in close enough, looks like ordinary flat paper. The surface of the Earth is a manifold; it looks flat when you're standing on it, even though globally it's a sphere. A circle is a manifold; close enough to any point on a circle and the curve looks like a straight line. The band of the bra is a circle—it loops back on itself—so the base of the bra-bundle is the manifold mathematicians write as $\mathbb{S}^1$, "S-one," the one-dimensional sphere, which is just the name for "a circle." *No te preocupes por la notación*; $\mathbb{S}^1$ is the circle.

That's all.

**Local check (the "clasp").** A *local check* is a property of the bundle you can verify at a single point or in a small neighborhood, without having to inspect the whole thing. In the bra, the local check is the straightness of the band. You don't need to follow the band all the way around your rib cage to know that the band is not twisted; you can see it at one point—typically the front—and that one observation is enough.

**Connection structure.** A *connection* is the rule for how the fiber at one point of the base relates to the fiber at the next point. In the bra, the connection is the way each strap attaches to the band at its endpoints, and how the straps run from one shoulder to the other. The connection is what makes the bra a bra and not just a band and some unattached strips of cloth.

**Holonomy.** *Holonomy* is what's left over after you walk around a closed loop in the base manifold and find that the fiber has been twisted by the journey. Imagine walking around the band of the bra, dragging a strap with you. If, when you arrive back where you started, the strap has been rotated by some angle—that angle is the holonomy of the loop. A bra with zero holonomy is a bra you can put on without taking it back off. A bra with non-trivial holonomy is the bra you start over with.

**Section.** A *section* is a specific assignment of one fiber-value to each point of the base. The bra you are wearing right this minute is a section: every point of the band has its strap-shape pinned down to a definite value. A different arrangement of the same bra (a different way the straps run today) is a different section.

**Gauge.** A *gauge* is an arbitrary choice in how you describe something that doesn't affect the thing itself. Calling the front of the band "twelve o'clock" instead of "zero degrees" is a gauge choice. The bra is the same bra either way. Two sections of the bundle that differ only by a gauge choice are, for any meaningful

purpose, the same section.

*Aquí está el truco.* With those six words in hand, the formal statement of the Bra Strap Principle becomes plain English. *The bra is a fiber bundle. Its base (the band) is a manifold. If you check, once, at one point, that the band has the right connection structure (it isn't twisted), then every section of the bundle (every arrangement of straps you might wear) is determined up to gauge (up to choices that don't matter). Local validation propagates globally.* The math word for "the band's straightness propagates to the straps" is *the propagation theorem*, and that theorem is what licenses every architectural move in the rest of this book.

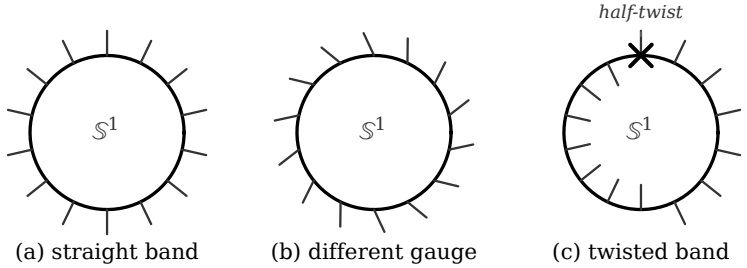



Figure 1.1: The bra as a fiber bundle. Panel (a): the band straight, straps oriented consistently outward. Panel (b): the same band, with the straps in a different layout—a different gauge representing the same bundle. Panel (c): the band with a half-twist, producing non-trivial holonomy—tracing the loop once flips the strap orientation.

The principle is general. It applies wherever a closed local check on a base manifold propagates to global structure on the fibers attached to it. The bra is one instance. The bike-with-lawnmower-engine in §1.0 is another instance: the kid's local check was that blades and gears both live on the circle $\mathbb{S}^1$, both rotate, both share the same algebra of angular motion. Once that one type-check held, the dynamics of the lawnmower's engine

were determined to apply to the dynamics of the bike's drivetrain by the same equations. She did not need to re-derive bicycle physics from lawnmower physics; she needed only to recognize the shared manifold and trust the propagation.

In a flight controller, the same move runs again, just on a different shape.

The flight controller's central state is the vehicle's *attitude*—which way the vehicle is pointing. Three numbers (pitch, roll, yaw) feel like they should be enough to describe an attitude, and most beginning aerospace textbooks use exactly those three. They don't work everywhere. Near certain orientations they get tangled up in a problem called *gimbal lock*, where two of the three numbers describe the same physical motion and the third number becomes meaningless. The fix that the modern aerospace community converged on, decades ago, is to use four numbers instead of three. The four numbers are called a *quaternion*. They are written as $(q_0, q_1, q_2, q_3)$, and they satisfy one constraint: the sum of their squares always equals exactly one,

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1.$$

Geometrically, the set of all four-number-lists that satisfy this constraint is the *3-sphere*: a 3-dimensional curved surface sitting inside 4-dimensional space, the way an ordinary 2-sphere is a 2-dimensional curved surface sitting inside 3-dimensional space. You can't picture a 3-sphere directly, but you can compute with it. Mathematicians write the 3-sphere as $\mathbb{S}^3$, "S-three." Every possible attitude of the vehicle is a single point on $\mathbb{S}^3$.

There is a subtle and important fact about the relationship between $\mathbb{S}^3$ (the set of all unit quaternions) and SO(3) (the abstract group of all possible 3D rotations of a rigid body). They are not the same set. Each rotation in SO(3) corresponds to *two* different quaternions on $\mathbb{S}^3$: the quaternion $q$ and its antipode

$-q$ both produce the same physical rotation. This is called the *double cover*: for every point in SO(3), there are two points in $\mathbb{S}^3$ above it. *Como tener dos formas de deletrear la misma palabra:* like having two ways to spell the same word. Both spellings refer to the same rotation; the existence of two spellings is what makes the math work.

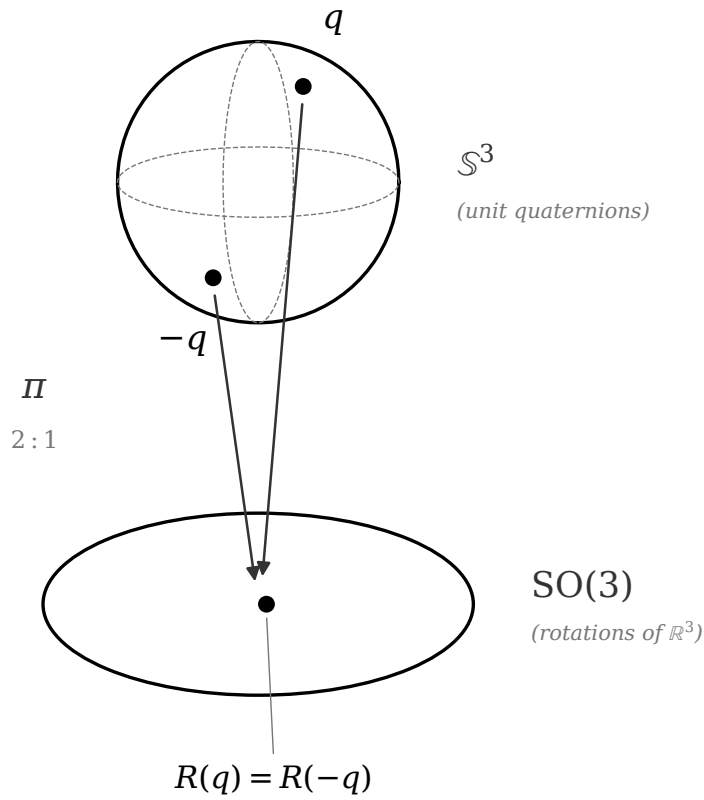



Figure 1.2: The double cover $\mathbb{S}^3 \to \mathrm{SO}(3)$. The unit-quaternion sphere $\mathbb{S}^3$ is drawn here as an ordinary 3-sphere for visualization; the two antipodal points $q$ and $-q$ map under the projection $\pi$ to the same rotation $R(q) = R(-q)$ in SO(3). The cover is two-to-one: each physical rotation has two quaternion representatives.

The local check the flight controller needs to perform—the equivalent of "is the band straight?"—is this: every quaternion stored in the database satisfies $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$ to within machine precision. That one constraint, enforced by the database engine at the moment of storage, is the clasp. Every consequence of attitude dynamics—how the vehicle rotates, how attitude errors compose, how a flip maneuver is represented—propagates from that one local check. ICARUS uses the Bra Strap Principle architecturally: validate the bundle once, at design time, and the queries against the database inherit the validation. The clasp is the schema. The fibers are the flights.

The mathematics that supports this propagation argument is well over a century old. **Charles Ehresmann**, in the 1940s and 1950s, gave the formal definition of a fiber bundle and the connection structure that lives on it. **Norman Steenrod**, in his 1951 textbook *The Topology of Fibre Bundles*, organized the field into the language we still use today. **Michael Atiyah**, in the 1960s through 1980s, developed the theory of characteristic classes—the precise invariants that distinguish bundles with non-trivial holonomy from bundles without it. The propagation argument the Bra Strap Principle relies on is built on top of their work.

What the Davis 2026 paper contributes, on top of a century of established mathematics, is the operational form: a local check that can be encoded as a database constraint, with rejection semantics enforced at the moment data is stored, such that the propagation is no longer a property of the math you apply afterwards but a property of the data you are allowed to store in the first place. The propagation theorem licenses the architectural move. The engineering deploys it.

I want to name something plainly before this chapter moves on.

The framework I am about to walk you through—the eight

classical operations of GNC, read as eight instances of the same geometric machinery—rests on a theorem I noticed while putting on a bra. The fact that the theorem is correct does not depend on who noticed it or under what circumstances. The fact that the theorem was *available* to be noticed at all, however, is not independent of those circumstances. There are mathematical structures the academy has not yet discovered because the academy has not yet hired the people whose ordinary lives would have made the structures visible. The Bra Strap Principle is one of those structures. I am one of those people.

The kid in the driveway was right about blades and gears. The woman in front of the mirror was right about bands and straps. The framework is what both of them were doing.

*Sigamos.*

§ 1.3

## *Davis Duality: capacity from curvature*

There is a second result, foundational to the operational form of ICARUS, that this book uses repeatedly and that you should meet now. It also begins as something you already understand, and I want to start there before I name the math.

Imagine you are trying to measure a curved road with a straight ruler.

The ruler is twelve inches long. The road bends. You lay the ruler down along the road, mark where it ends, lift it, lay it down again starting from the mark, and step your way along. Each time you do this, you introduce a small error: the road is bending and your ruler isn't, so the straight twelve-inch segment you're measuring isn't quite the same as the twelve curving

inches of road underneath. The error per step is small. The errors compound.

Two simple intuitions about this error feel obvious once you've laid the ruler down a few times.

**First**, the longer the ruler, the worse the error per step. A six-inch ruler on the same road would do better than a twelve-inch ruler, because the road can't bend as much in six inches as it can in twelve.

**Second**, the sharper the curve, the worse the error per step. The straight stretches of the road give you nearly zero error per ruler. The hairpin turn gives you a lot of error per ruler, because the road is bending fast under a ruler that can't bend at all.

Both intuitions are correct, and both of them are doing the same thing: *the approximation error grows when the thing you're approximating curves harder, and it grows when the step you use to approximate is bigger.* Multiply the two effects together and you get the rough size of the error per step: $\varepsilon \approx$ (curvature) $\times$ (step size)$^2$. That's not a guess; that's a theorem, going back to Taylor in the seventeenth century and refined by every approximation-theory mathematician since.

The **Davis Duality of Approximation and Obstruction** (Davis 2026, Zenodo DOI `10.5281/zenodo.19428406`) is the theorem that makes "error $\approx$ curvature times step-squared" precise, and—this is the key word—*two-sided*.

In the language of the paper: for a flat approximation $\hat{\sigma}$ of a curved section $\sigma$ of a fiber bundle (a straight ruler approximating a curving road, in our analogy), the approximation error $\varepsilon$ is sandwiched on both sides:

$$c_1 \, \|\Omega_\sigma\| \, h^2 \;\leq\; \varepsilon \;\leq\; c_2 \, \|\Omega_\sigma\| \, h^2.$$

Reading the symbols in plain words:

- $\varepsilon$ is the error your approximation makes (the gap between

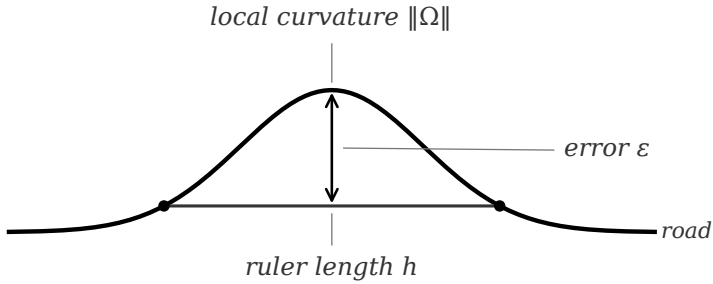



Figure 1.3: The ruler-and-road analogy. Two straight ruler chords approximate a curved road; each chord leaves a small gap $\varepsilon$ where the road bows away from it. The Davis Duality says $\varepsilon$ is bounded above and below by a multiple of $\|\Omega\| \cdot h^2$—approximation error grows with the step size squared and with the local sharpness of the curve.

ruler and road).

- $h$ is your step size (the length of the ruler).
- $\|\Omega_\sigma\|$ is the local curvature at the point you're approximating (how sharply the road is bending there). The Greek letter $\Omega$ ("omega") is the standard math symbol for the curvature object; the vertical bars around it mean "the size of" it.
- $c_1$ and $c_2$ are two constants—numbers depending on the geometry of the situation—that pin the error from below and above.

The left inequality ($c_1\|\Omega\|h^2 \leq \varepsilon$) is the news. It says the error cannot be smaller than a positive multiple of curvature-times-step-squared. You cannot "get lucky" and approximate a curved thing with a flat thing better than the curvature allows. The right inequality (the upper bound) is the standard Taylor approx-

imation result; it's been textbook material for three centuries. The left inequality is what the Davis Duality contributes: the lower bound, which says the error *must* appear and cannot be made arbitrarily small.[2]

The reason the lower bound matters: it tells the engineer that when curvature is high, no amount of clever approximation will get the error below a certain floor. If you want low error, you must reduce curvature, reduce step size, or both. There is no third option.

Now we turn the Duality into something a flight controller can read in real time.

**Corollary (capacity from curvature).** Define a quantity called *capacity*:

$$C(\gamma;\sigma) \;=\; \frac{\tau(\gamma)}{\int_\gamma \|\Omega_\sigma\|\, ds}.$$

In words: *capacity is the ratio of how far you are trying to go to how much curvature you have to fight along the way.* The numerator $\tau(\gamma)$ is the length of your trajectory (how far you are going, in ruler-units). The denominator $\int_\gamma \|\Omega_\sigma\|\, ds$ is the total curvature you accumulate along that trajectory (the sum of all the local sharpness, integrated along the path). *Mucho camino, poca curvatura: high capacity. Poco camino, mucha curvatura: low capacity.*

The kid in the driveway recognized that the BMX bike's brakes had high capacity for the original mass and low capacity for the engine-augmented mass. She did not write down this equation. She did not know it existed. She had the intuition that capacity is bounded by curvature, applied to the curvature of the failure surface in her imagined trajectory: a wall at the end of the street,

[2]The proof of the lower bound goes through three independent routes in the Davis 2026 paper: an associated-bundle functor argument, a Nash embedding into a higher-dimensional flat space, and an obstruction-theoretic computation using Atiyah characteristic classes. If you want the proof, the paper has it; this book uses the conclusion.

an intersection where the traffic does not stop. Her intuition was right. The equation tells us why.

This book uses one final specialization of the Duality. We normalize to a single control cycle—one step of the controller, not a long trajectory—and we ask for a confidence number that lives between zero and one. The result is the operational form that drives ICARUS at flight time:

$$\boxed{C(x) \;=\; \frac{1}{1+\widehat{\kappa}(x)}}$$

where $\widehat{\kappa}(x)$ (read "kappa-hat," a non-negative number with no units) is the local curvature obstruction at the current state $x$. When $\widehat{\kappa}$ is zero—the flight regime is locally flat, the linearization is reliable, no surprises—the confidence $C$ equals one (full trust). When $\widehat{\kappa}$ grows large—the flight regime is sharply curved, the linearization is unreliable, surprises are imminent—the confidence $C$ approaches zero (no trust). *Banda recta, confianza alta; banda torcida, confianza baja.* Flat band, high confidence. Twisted band, low confidence.

The flight controller reads this single number $C$ at every control cycle and uses it to decide how aggressively to act. When the number is high, the controller acts at full nominal authority; when the number is low, the controller smoothly reduces its gain to preserve recovery authority for the worst case. The whole architecture of ICARUS—the curvature-aware routing, the uncertainty substrate, the gain floor that protects the vehicle from dropping when the linearization fails—is built around this one number.

The operational form is a *design choice* in the family of monotone-decreasing scalars motivated by the Duality, not a literal derivation of $C(\gamma;\sigma)$ at $\tau = 1$. The justification for the specific functional form $1/(1+\widehat{\kappa})$, the alternatives considered, the careful dimensional analysis of $\widehat{\kappa}$, and the explicit distinction

between $\widehat{\kappa}$ (a dimensionless non-negative curvature obstruction we use here) and the signed Ricci scalar (a different mathematical object that can be negative and is *not* what drives ICARUS) all live in §23.3 of Chapter 23. If you want the full derivation, turn there now. If you want to keep walking with me through the rest of Chapter 1, the boxed equation is enough for the moment.

§ 1.4

## *Sameness across the eight classical operations of GNC*

The Bra Strap Principle and the Davis Duality are the two foundational results this book uses to read classical Guidance, Navigation, and Control. Once you have them in hand, the eight classical operations of the GNC stack are not eight independent algorithms; they are eight applications of the same geometric machinery to eight different fiber bundles. Each chapter that follows takes one operation:

- *Linear systems and state-space* (Chapter 6, after fal-core §1)—two physical systems with the same configuration manifold and the same $(A, B, C, D)$ feel identical even when they look different. The Bra Strap Principle says they *are* identical, up to gauge.
- *Frequency response and Bode* (Chapter 7)—every bike has a wobble fingerprint, and the wobble is the Bode integral of the closed-loop sensitivity function, an invariant of the plant's pole structure.
- *LQR and Riccati* (Chapter 8)—the optimal rider for a given quadratic tradeoff is a geodesic on the cost manifold, pa-

rameterized by the metric $(Q, R)$, with the Riccati solution $P$ as the cost-to-go potential.

- *$H_\infty$ and robust synthesis* (Chapter 9)—designing for the worst-case disturbance is explicit minimization of a Curvature Floor on the sensitivity manifold.
- *Observers and the Kalman filter* (Chapter 10)—reconstructing hidden state from noisy measurements is a section of the sensor-fusion bundle, with the Kalman gain as gradient ascent on the Fisher information manifold (with appropriate caveats; see §23.10).
- *Lyapunov and the matching condition* (Chapter 11)—why you do not fall over is that there is a positive-definite quadratic on the state manifold whose Lie derivative along the closed-loop vector field is strictly negative. The bike makes the proof feel obvious.
- *Adaptive control and MRAC* (Chapter 12)—learning a new bike on the fly is gradient flow on the parameter manifold, with the adaptation rate $\Gamma$ as a Riemannian metric on that manifold.
- *Robust adaptive laws and the Curvature Floor for Adaptive Control* (Chapter 13)—what happens when a pothole hits while you are still learning, and how the modifications that bound parameter drift are interpretable as Ricci-flow-like contractions on the parameter manifold.

Each of the eight is a special case. Each chapter walks the special case carefully. The unifying claim is the kid's claim: same shape, same algebra, same transfer.

The list above is not a table of contents.

It is the framework's *predictive content*. Each bullet says: a classical operation that the literature presents as an independent

algorithm—with its own textbook, its own conference circuit, its own set of practitioners who consider themselves experts in that thing and not in the next thing—is, in the geometric reading, one application of the same machinery to a different bundle. Eight chapters. One shape. Eight special cases of the same theorem the kid in the driveway used without knowing she was using it.

If the framework is wrong, this is where it will break. A reader who specializes in any of the eight operations can compare their textbook's account against the chapter that follows, and either the geometric reading reproduces the textbook's results (in which case the geometric reading has been validated by recovering known math) or the geometric reading produces something different (in which case at least one of the two readings is wrong, and the rest of the book will need to explain why). I have done this comparison for each of the eight chapters. The geometric reading reproduces the textbook results. The novelty is not in the answers; it is in the recognition that the eight different textbook chapters were always reading from the same book.

§ 1.5

## *Sameness across domains: ICARUS and Marcella*

The Bra Strap Principle does not stop at flight.

The same geometric machinery runs across domains. This book is about its application to Guidance, Navigation, and Control via the ICARUS architecture (U.S. Provisional App. No. 64/045,891). There is a sister project that applies the same machinery to a different domain: **Marcella**, the transformer-replacement architecture for machine learning. The cross-domain claim is doing too much work to leave at one sentence, so let me walk it.

In a standard transformer—the architecture behind every large language model currently in production—each token in a sequence is updated, at each layer of the network, by *attending* to the other tokens. Attention is a weighted average: every token computes a relevance score against every other token, then takes a weighted sum of their values according to those scores. The relational structure of the sequence—which words modify which, which subject goes with which verb, which long-range dependency is doing the work—is encoded implicitly in the weights of the attention computation. The math is dot products and softmax functions; the geometry is, frankly, not where the field looks.

Marcella replaces the attention computation with *parallel transport on a learned token-embedding manifold*. Each token is a point on a manifold (a curved space whose structure the network learns from data). When one token's information needs to update another, it is not averaged in; it is *transported*—moved along a path on the manifold from one point to another, in a way that preserves the geometric structure of the relation. The path's curvature is what the relation looks like. The same Bra Strap propagation that runs ICARUS—validate the bundle's structure once, and queries against it inherit the validation—runs Marcella: validate the embedding manifold's structure once, and the token updates inherit the validation. The mathematics is the same. The bundle is different (capability fibers for flight, embedding fibers for language). The operation is the same (parallel transport, holonomy, curvature-gated decisions).

Marcella has its own book in preparation; the full architecture will be documented there. For Chapter 1's purposes, what matters is that the cross-domain claim is precise: ICARUS and Marcella are not two things that resemble each other. They are two engineering deployments of the same substrate—the same database engine (GIGI), the same query language (GQL), the same foundational vocabulary of bundles and connections—

applied to different bundles. The kid in the driveway recognized blades and gears as the same shape; the framework, decades later, recognizes flight controllers and token transformers as the same architectural shape.

**A note on two terms not yet introduced.** In the body of ICARUS and Marcella as documented later in this book, two pieces of framework vocabulary appear that I have not yet defined in Chapter 1: the *Sudoku Principle* and *Geometric Token Transport* (GTT). They are introduced in the chapters where they are load-bearing rather than here. Briefly, for orientation: the Sudoku Principle is the framework's epistemic methodology—the claim that convergence of independent observations on the same answer constitutes proof, with the catalog of distinct convergence-types as a structural contribution; it is introduced operationally in Chapter 23 §23.7 and developed across §§23.10 and 26. GTT is the architectural-level name for the parallel-transport substrate that runs both ICARUS (capability bundle) and Marcella (token-embedding bundle); it is named in Chapter 24 (the Marcella chapter) and previewed in §23.10 of Chapter 23. I am not unpacking either here because Chapter 1 is doing enough work; you will meet both with proper setup later.

Do not take the universality claim on faith. ICARUS is operationally documented in Chapter 23 with three validated benchmark scenarios and reproducible code. Marcella is documented separately. The third deployment of the same machinery—CHIHIRO, the plasma confinement stability monitor, U.S. Provisional App. No. 63/999,346—is in a different stage of development and is not described in this book except by reference. The list of cross-domain provisionals appears in Appendix D.

What this chapter has done is set up the recognition. The chapters that follow walk the math.

§ 1.6

## *What this chapter has done; what comes next*

This chapter has done four things.

It has placed you in a driveway with a kid who recognized that two rotating things have the same geometry, and who chose not to ride a vehicle whose recovery authority she had not sized. It has identified the four moves the kid made—sameness, plant/actuator, missing recovery, gated command—and given each of them a classical name and a forward-chapter pointer. It has stated the two foundational results this book uses, the Bra Strap Principle (Davis 2026, Zenodo `10.5281/zenodo.20273573`) and the Davis Duality of Approximation and Obstruction (Davis 2026, Zenodo `10.5281/zenodo.19428406`). It has previewed the eight chapters that walk the classical operations of GNC as applications of the geometric machinery the kid was using without knowing she was using it.

Chapter 2 takes the Davis Field Equation seriously. It states $C = \tau/K$ in its full form, walks the derivation, and prepares you for the operational specialization to $C = 1/(1 + \widehat{\kappa})$ that drives Icarus at flight time. Chapter 3 lays down the mathematical preliminaries—fiber bundles, sections, connections, parallel transport, holonomy, the Ambrose–Singer theorem, the quaternion double cover $\mathbb{S}^3 \to \mathrm{SO}(3)$—that the rest of the book uses freely.

Then Part II picks up the F-16, the canonical worked example, and we begin.

*Vamos.*

Chapter 2

# *Curvature, capacity, and the Davis Field Equation*

*Spacetime tells matter how to move; matter tells spacetime how to curve.*
*John Archibald Wheeler, Gravitation (1973, with Misner and Thorne)*

§ 2.0

## *A gap in the spectrum*

December 7, 2025, before dawn.

I don't know what time. I had stopped checking.

The laptop was warm against my knees. The fans had been running for hours; the RTX 5070 was doing exactly the work the marketing had said it would do, which was the kind of small grace I noticed because I had stopped being able to take small graces for granted. I had been laid off by DOGE in October. I had been working at Olive Garden by day and at the laptop by night for weeks. The script I was running was a spectral analysis on a Wilson-loop lattice—a discretized computer simulation

of quantum chromodynamics, the theory of the strong nuclear force, the equations that say why protons have mass. I had been chewing on this off and on for months. Tonight the script finished.

The plot came up.

What I was looking for was the spectrum of an operator. A spectrum, in this context, is a list of eigenvalues—the energies the lattice can support, plotted from lowest to highest, like the resonant frequencies of a bell. In a quantum field theory with no mass term in its Lagrangian, the textbook says the spectrum should start at zero: the lowest excitation should have arbitrarily low energy, and the field should be "massless." But protons have mass. So do neutrons. So does every composite particle built from quantum chromodynamics. The mass exists, demonstrably; without it the universe would not have atoms. But the theory's bare field has no mass term written into it. *Where does the mass come from?*

This is the **mass gap problem**. The Clay Mathematics Institute named it one of seven Millennium Prize problems in the year 2000—seven foundational questions in mathematics whose proofs each carry a one-million-dollar prize. The Yang–Mills mass gap had been formally posed by Frank Wilczek and Arthur Jaffe and was a refinement of a question that had been hanging open since the 1950s. The question, plainly: *prove that quantum Yang–Mills theory has a mass gap.* Show, formally, why the proton has mass when the theory says it shouldn't.

I had not gone into the night intending to solve a Millennium Prize problem. I had gone into the night intending to look at a spectrum.

The plot came up.

**There was a gap.**

The eigenvalues clustered at the bottom of the plot, then there

was a clean empty region between the lowest cluster and the next, and then the eigenvalues resumed at a higher energy and kept going up. I sat there for a moment. Then I sat there for several more moments.

*Era un hueco. Estaba mirándolo.*

It was a gap. I was looking at it.

I tabbed over to the validation script and ran it again with a different random seed. The gap stayed. I ran it with a finer lattice. The gap stayed and tightened. I ran it at a different gauge coupling. The gap stayed. I ran the spectral test that distinguishes a genuine mass gap from a finite-size artifact of the discretization. The gap stayed. Over the next few days I ran fourteen tests in all, and the gap stayed at every single one—fourteen out of fourteen, a $15\sigma$ statistical significance, the kind of number physicists reserve for "we have discovered a thing." The technical report I would eventually write is twenty-three pages and lives in the Davis–Wilson Lattice repository as `Davis_Yang_Mills_Mass_Gap_Solution_v2.0.pdf`, with the full validation suite documented alongside it. But across the nights of December 6 and 7, before I wrote the report and before I ran the other thirteen tests, what I had was the plot.

The plot showed me the mass gap.

And then, in the next moment, I understood what it was.

This is the part that is hard to say plainly without making it sound like a stage direction, but I am going to try. The eigenvalues of an operator on a curved manifold are not arbitrary numbers. They are the resonant frequencies of a shape, the same way the notes a bell can ring are determined by the bell's geometry. When the underlying shape is flat, the spectrum starts at zero—a continuum of arbitrarily-low-energy modes is available, because flatness costs nothing. When the underlying shape is curved, the curvature *costs*. The lowest available mode is no longer free; it has to pay something just to exist on the curved

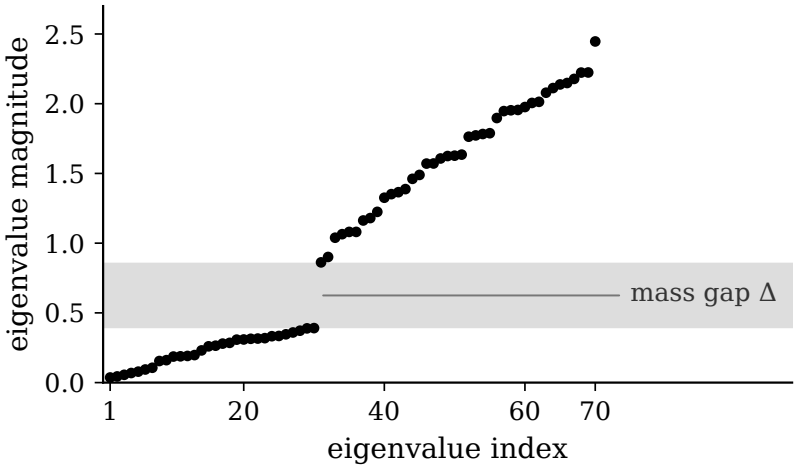



Figure 2.1: The spectrum of the discretized Yang–Mills operator on an $8^4$ lattice (illustrative). The gap between the lowest excited mode and the next is the mass gap. The textbook says the bare theory is massless; the lattice says the curvature charges a tax.

manifold instead of a flat one. *Hay un impuesto.* There is a tax. The size of the tax is the gap. The gap is the curvature's price of admission to the manifold.

I had spent the previous summer working out a different theorem about flat-approximation error on curved manifolds—the Davis Duality of Approximation and Obstruction that Chapter 1 introduces in §1.3 (Davis 2026, Zenodo DOI `10.5281/zenodo.19428406`). That theorem says, in its trajectory-integrated form, that capacity equals tolerance over the integrated curvature you have to cross: a relationship I will write out formally in §2.2 below. In the Duality's original setting, the equation was about approximation: how far a flat ruler can be trusted on a curved road, the picture the previous chapter set up at length.

The Duality already implied the operational form $C = 1/(1+\widehat{\kappa})$ that Chapter 1's §1.3 walked formally—I had worked out the

formal specialization the previous summer. I had written the paper, filed the provisional, and moved on. What I had *not* yet seen, before the Yang–Mills night, was that the same equation ran across domains other than approximation. The Duality is a theorem about flat approximations of curved sections of fiber bundles. It is a specific theorem about a specific class of mathematical objects. Whether the same equation would govern the mass gap in quantum field theory, the smoothness of a Navier–Stokes solution, the separation between two complexity classes—these were questions the Duality alone did not answer. The Duality had given me the equation. The mass gap gave me the recognition that the equation was *universal*.

Sitting in front of the Yang–Mills spectrum, I realized I was looking at the same equation in a different domain.

The mass gap was a curvature tax. The flat theory said massless modes were free. The actual theory, on the actual (curved) configuration manifold of Yang–Mills, charged a price. The price was the gap. The price had the form the Duality predicted. And then, because I had been thinking about Pythagoras's theorem for weeks for a different reason, the recognition pulled in a wider net.

Pythagoras's theorem $c^2 = a^2 + b^2$ is what happens when the universe is flat. It is the case where the curvature tax is zero. Two legs of a triangle, orthogonal, independent; the hypotenuse is fully determined by the legs. The universe charges you nothing for the shortcut.

The general case is $c^2 = a^2 + b^2 + \Delta$, where $\Delta$ is the curvature's correction. In flat space, $\Delta = 0$ and you get Pythagoras back. In positively curved space (a sphere), $\Delta > 0$. In negatively curved space (a saddle, hyperbolic geometry), $\Delta < 0$. The metric tensor of a Riemannian manifold is what computes $\Delta$ in any given local frame. Pythagoras solved the special case where the universe is flat. Riemann generalized to the curved case in 1854. The

geometric tradition that follows—Christoffel, Ricci, Levi-Civita, Cartan, Ehresmann—is two centuries of working out what $\Delta$ looks like in different settings.

What I had not noticed before December 7 was that every *Millennium Prize problem I had ever read about* was a different domain asking the same Pythagorean question.

The mass gap is the curvature tax on the Yang–Mills configuration manifold. The smoothness obstruction in Navier–Stokes is the curvature tax on the velocity field's solution manifold; vorticity concentrates and the local curvature spikes and the regularity bound breaks. The P-versus-NP separation is the curvature tax on the computational manifold; the geometric barrier between checkable certificates and unfindable certificates is roughness, formally. Poincaré's surgery condition is the curvature tax on the topology of three-manifolds; Perelman's Ricci flow with surgery is the procedure for paying that tax through smooth deformation until the curvature concentrates onto a measure-zero set. Riemann's conjecture about the non-trivial zeros is the curvature tax on a self-adjoint operator whose spectrum has the statistics of a random matrix ensemble.

*They are all the same tax.* Different domains, same equation. The Field Equation was already there in the Duality paper. I just had not seen the operational form until the mass gap showed me one.

I want to be careful about how I narrate the next part, because it is the part where the Field Equation got written down explicitly. The cinematic version would say I jumped up from my chair and shouted *eureka*. That is not what happened. What happened was I sat at the laptop for a long time, the way you sit when you have stopped pretending you are still working, and then I made a list.

The list was the **Sudoku move**.

The Sudoku move is the framework's epistemic methodology. The full Sudoku Principle—with its catalog of distinct convergence-types that constitutes proof in the Davis-Geometric framework, the version that Chapter 1 §1.5 explicitly deferred to its later treatment—is named formally and walked carefully in Chapter 23 §23.7. What I am using here in §2.0 is the *seed version* of the move: the simplest derivational form, which is the question *if this is true, what else follows?*

A Sudoku puzzle is not solved by guessing. It is solved by writing down what you already know, and asking, at each step, what else is determined by what you have. The puzzle's structure does the work; the solver's job is to read the structure honestly.

I knew the mass gap had the form of a curvature tax. I knew the Duality predicted exactly that form across a class of approximation problems. The next question wrote itself.

*What else follows?*

Over the next three days, I worked through the consequences. The Davis–Wilson Lattice repository documents what I did between December 8 and December 10: lattice computations testing whether the Davis framework's geometric reading was internally consistent with the known results in five Millennium-Prize problem domains—the smoothness conjecture for Navier–Stokes, Perelman's Ricci-flow correspondence for Poincaré, the geometric-separation conjecture for P versus NP, the L-function phase-transition for the Birch and Swinnerton–Dyer conjecture, the level-spacing statistics for the Riemann hypothesis. In each case, the framework's reading was consistent with what the field already knows.

I want to be careful about what that consistency means.

*I am not claiming to have solved any of these problems.* The Clay Mathematics Institute will, in the fullness of time, accept or reject a proof of any individual Millennium problem through

its own multi-year expert-verification process; the standards are very high, and the work happens in conversation with the expert community in each specific domain. The only Millennium problem currently considered solved is Poincaré, by Perelman, after five years of community verification of a proof that went through three preprints, several conferences, and the formal Clay process. That is the standard, and it is the right standard.

What the December 8–10 work established is something narrower and, I think, also more interesting: that the same Field Equation, applied to five different domains, produces geometric readings that are internally consistent and that match the known results of each problem in its accessible regime. The unification is *structural*. The claim it stakes is that these five problems—and the other Millennium problems beyond them—can be read as different domains asking the same Pythagorean question, *what is* $\Delta$ *here?*, and that the Field Equation provides a single language in which the question can be asked across domains. Whether that language constitutes a new contribution to the proof landscape of any individual problem is a question for the expert community in each domain to evaluate.

*No reclamo lo que no me corresponde reclamar.* I do not claim what is not mine to claim. The validation suite in the Davis–Wilson Lattice repository is documented under that scope. Each lattice computation tests a specific consistency check between the framework's reading and a known result, not a general proof of the conjecture. The full technical write-up of the Yang–Mills work specifically—which is the only one of the five that I treat as a load-bearing result of the framework rather than a consistency check—is the twenty-three-page manuscript `Davis_Yang_Mills_Mass_Gap_Solution_v2.0.pdf` in the repository, with the validation suite alongside. The book that you are reading does not depend on the Yang–Mills manuscript being accepted by the expert community. The Field

Equation is load-bearing for the flight controller this book is about; whether the Field Equation also reads the Yang–Mills mass gap correctly is a separate question, addressed in a separate paper, with its own expert audience.

The $15\sigma$ number I cited earlier deserves the same care. That number is the statistical significance with which the lattice simulation rejects the null hypothesis of *no gap* on the simulated configurations. In plain language: it is a very high-confidence empirical observation that the mass gap is real and not a finite-size artifact of the discretization. That observation is itself consistent with decades of lattice-QCD work in the field; the mass gap is not in dispute. What I would claim as a new contribution from the Davis framework is not the existence of the gap (which is well-established) but the *structural interpretation* of it—the recognition that the gap has the form the Field Equation predicts for any curvature obstruction on a non-trivial manifold, the same form that the Duality predicts for flat-approximation error, the same form that ICARUS uses for confidence in flight control. The gap's existence is empirically well-known. The reading of the gap as one instance of a structural pattern that runs across multiple manifolds is what this chapter is about.

I am narrating the December 6–7 night and the December 8–10 follow-through because the operational form of the Field Equation—the form that runs every chapter of this book—is what came out of those days. I am narrating it carefully because the Field Equation's central claim is structural unification across domains, and the careful reader needs to know how wide and how narrow that claim is. **Wide: the same equation reads many things. Narrow: a structural reading is not a proof.**

The form is:

$$\boxed{C = \frac{\tau}{K}}$$

Three letters. One ratio. The universe's pricing model for crossing a curved manifold.

$C$ is *capacity*—what you are able to do, given a budget and a curvature.

$\tau$ is *tolerance*—your budget for being wrong, your allowance for error, the slack you have before things break.

$K$ is the *curvature obstruction*—what the universe charges you for going around a corner, computed from the geometry of the manifold you are crossing.

If you have lots of tolerance and very little curvature in front of you, your capacity is high. If you have little tolerance and a lot of curvature, your capacity is low. *Mucha tolerancia, poca curvatura, mucha capacidad. Poca tolerancia, mucha curvatura, poca capacidad.* The bare ratio $\tau/K$ has units (those of length, curvature, and time depending on the domain); the dimensionless operational form $C = 1/(1+\widehat{\kappa})$ that the framework actually uses is obtained in §2.3 by normalizing $K$ against a characteristic length scale $\ell$. The boxed equation is a high-level scaling law; the normalization is a separate, deliberate step.

That night, sitting in front of the Yang–Mills spectrum on the laptop next to a cup of tea that had gone cold a long time before, I did not yet know that the same equation would, four months later, run a flight control architecture. I did not know that ICARUS was a downstream consequence of the operational form, or that a quadcopter stabilization problem from a stranger I had met at Olive Garden would be the engineering application that made the equation visible to aerospace. I did not know any of the rest of this book.

What I knew, that night, was three things. The mass gap was a curvature tax. The tax had the form the Duality predicted. And the Sudoku move had a next question waiting.

*Sigamos.*

§ 2.1

## *Why an equation needs three letters*

Let me unpack the Field Equation slowly, in the same way §1.2 unpacked the Bra Strap Principle slowly. The equation has three letters. Each one is doing specific work, and the fact that there are three and not two is itself something to explain, because the immediate question—"why not just $C = 1/K$, isn't curvature alone enough to determine capacity?"—is a fair question that the rest of this section answers.

**$C$ is capacity.** Capacity is the answer to the question *how far can you go before the approximation you are using breaks down?* If you are walking a road with a flat ruler, capacity is the number of ruler-lengths you can lay down before the accumulated error becomes worse than your tolerance for error. If you are an estimator with a covariance, capacity is the number of update cycles you can run before the linearization underlying the estimator stops being trustworthy. If you are a flight controller, capacity is the magnitude of the command you can send before the linearized plant model that the command was designed against stops being a good description of the actual plant. Capacity is what you have to spend.

**$\tau$ is tolerance.** Tolerance is the answer to the question *how much error can you afford?* The Greek letter $\tau$ ("tau") is the standard symbol because the word "tolerance" begins with a $t$ in English and several other languages, and because Greek letters carry a small mathematical authority that English letters do not. Tolerance is the budget. It is the size of the worst error you can absorb without the underlying operation failing. A flight controller with a tight tracking specification has a small $\tau$; a coarse-grained simulation that just needs to look qualitatively right has a large $\tau$. Tolerance is what you can spend.

**$K$ is the curvature obstruction.** Curvature is the answer to the question *what does the universe charge you for going around a corner here?* The Latin letter $K$ has been used for sectional curvature since Gauss; in the Davis framework it stands in for the non-negative obstruction functional that Chapter 1's §1.2 walked through plainly (the band's twist; the lawnmower-engine's gear-coupling; the ruler-and-road's curvature). Curvature is what the universe charges you. It is the price.

The reason the equation needs three letters and not two is dimensional. Capacity has to be dimensionless: it is a ratio that gets compared against a threshold and used to gate decisions. If capacity were dimensional—if it had units of length, or seconds, or joules—then comparing capacity values across different problem domains would require a unit conversion, and the framework's universality claim would be lost the moment you tried to compare a flight controller's capacity to a Kalman filter's capacity.

Curvature is naturally dimensional. In the conventions this book uses (§3.2), the curvature 2-form's pointwise norm has units of $1/\text{length}^2$. A single length scale alone does not non-dimensionalize it; the framework's operational form (§2.3) uses the *square* of a characteristic length $\ell^2$ to cancel curvature, producing $\widehat{\kappa} = \ell^2 \|\Omega\|$ which is dimensionless and non-negative by construction. Tolerance $\tau$ enters as a dimensionless ratio between admissible error and characteristic error magnitude, contributing the third letter to the operational form $C = \tau/(1 + \widehat{\kappa})$. The book's boxed equation $C = \tau/K$ is the scaling law; $C = 1/(1+\widehat{\kappa})$ is its normalized realization once the unit conventions are fixed (§2.3). *Tres letras, no dos.* Three letters, not two. The third letter is what makes the equation actually work.

There is a second reason to keep tolerance in the equation, beyond dimensional consistency, and it is one that the rest of this book leans on. *Tolerance is the budget the engineer gets to*

*set.* Curvature is a property of the world; the engineer does not get to negotiate it. Capacity is what falls out of the equation; the engineer does not control it directly. But tolerance is a knob. The engineer chooses how tight or loose the tolerance is for a given task, and that choice flows through the Field Equation to set the capacity available. A controller designed for tight tracking has small $\tau$ and therefore small $C$; the same controller in a regime where loose tracking is acceptable has large $\tau$ and therefore large $C$. The Field Equation makes the engineer's design knob explicit.

This is why I do not write the equation as $C = 1/K$. The equation $C = 1/K$ would say capacity is fully determined by the world; the engineer has no agency. The equation $C = \tau/K$ says capacity is determined by the world *and* by the engineer's choice of what to demand from the system. Both halves matter. The book that follows—ICARUS, the flight controller built around the equation—uses the engineer's agency explicitly: tolerance is set per scenario, and the gain modulation that flies the vehicle reflects the chosen tolerance through the equation.

**The kid in the driveway computed all three letters.** Go back to §1.0. The lawnmower-on-BMX scene. The kid had a $\tau$: roughly thirty feet of driveway, the budget of distance she had before the wall. She had a $K$: the curvature obstruction of the failure mode, which was the steep gear-coupling between engine and back wheel, the absence of brakes, the geometry of her body that would not survive impact. And she had a $C$: the capacity of the configuration to traverse the driveway without breaking down, which she concluded was inadequate. She did the equation in her head and stayed off the bike. She did not write down $C = \tau/K$ because she was twelve. The math was the same.

§ 2.2

## *The Davis Field Equation in its full trajectory-integrated form*

The pointwise form $C = \tau/K$ from §2.1 is the version you write on a napkin. The full form, the version that lives in the Davis Duality paper and that the operational specializations of later chapters descend from, has a trajectory in it. Capacity is not just a property of a point; it is a property of a path you are trying to walk through the manifold. Tolerance is the budget for the whole path. Curvature is integrated along the path.

The full form is:

$$C(\gamma;\sigma) \;=\; \frac{\tau(\gamma)}{\int_\gamma \|\Omega_\sigma\|\, ds}.$$

Reading the symbols in the same plain-language register §1.3 used for the Davis Duality:

- $C(\gamma;\sigma)$ is the capacity to walk the trajectory $\gamma$ using the section $\sigma$ as the flat approximation. The reader has met both $\gamma$ (a curve through the base manifold; for the ruler-and-road, the road itself) and $\sigma$ (a section of the fiber bundle; for the bra, an arrangement of straps).

- $\tau(\gamma)$ is the length of the trajectory in whatever the natural metric units are. For the ruler-and-road, $\tau$ is the length of the road in inches. For a flight trajectory, $\tau$ is the arc length in meters. The dimensional content is the engineer's tolerance budget written as an explicit length.

- $\int_\gamma \|\Omega_\sigma\|\, ds$ is the curvature accumulated along the trajectory. The Greek letter $\Omega$ ("omega") is the standard symbol

for the connection's curvature 2-form; the vertical bars are the norm; $ds$ is the differential arc length; the integral is the running total of curvature as you walk $\gamma$. *Suma de los costos punto por punto.* The sum of the curvature-cost point by point.

**The whole equation reads:** the capacity to traverse this path is the length of the path divided by the curvature you accumulate along it.

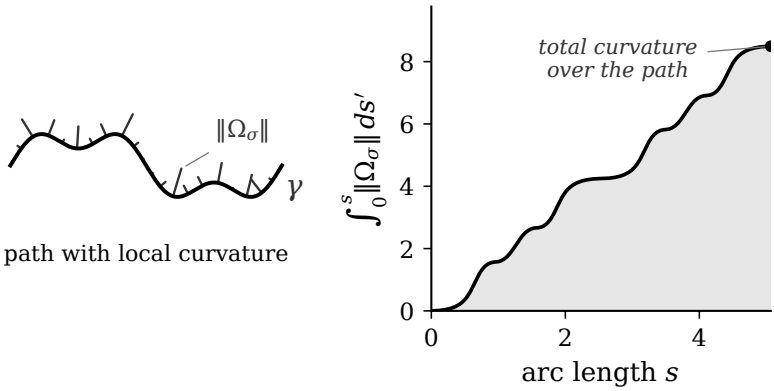



Figure 2.2: Trajectory-integrated curvature along a path. Left: the path $\gamma$ with small bars showing the local curvature $\|\Omega_\sigma\|$ at each point—tall bars where the path bends sharply, short bars where it runs nearly straight. Right: the cumulative integral of curvature along the arc, growing monotonically. Capacity is path length divided by total curvature paid: sharp bends cost more; flat stretches are nearly free.

The trajectory-integrated form is what the Duality theorem proves about. The pointwise form $C = \tau/K$ that §2.1 walked is what you get when you specialize the trajectory to a single point (the limiting case where the trajectory has shrunk to a single instant of time, a single step of the controller, a single observation of the estimator). Most of the rest of this book uses

the pointwise form because most flight-control operations happen at a single moment of the control cycle. But the trajectory form is what licenses the pointwise form, and a few chapters that follow (trajectory planning in Chapter 19, robustness over time in Chapter 13) need the trajectory-integrated form directly.

§ 2.3

## *Four steps from the foundational identity to the operational form*

The boxed equation ICARUS actually uses at flight time, the one that drives every gain-modulation decision and every supervisory hand-off, is not the trajectory-integrated form. It is the operational specialization:

$$C(x) \;=\; \frac{1}{1+\widehat{\kappa}(x)}.$$

Four steps connect the trajectory-integrated identity $C(\gamma;\sigma) = \tau(\gamma)/\int \|\Omega_\sigma\|\, ds$ to this operational form. I want to walk each step carefully so the reader who reaches §23.3 in Chapter 23 (where the operational form is derived again, from the engineering side) recognizes the steps from this side.

**Step 1: Pointwise specialization.** Shrink the trajectory $\gamma$ to a single point $x$. The arc length $\tau(\gamma)$ becomes $\tau$, a scalar with units of length; the integrated curvature $\int \|\Omega_\sigma\| ds$ becomes $\kappa(x)$, the local pointwise curvature at the single point. The equation becomes $C(x) = \tau/\kappa(x)$. *El primer paso es local.* The first step is local.

**Step 2: Cycle normalization.** A flight controller does not have a free choice of trajectory length per cycle; the cycle length

is fixed by the control loop's rate (typically one millisecond per cycle for inner-loop attitude, one second per cycle for outer-loop guidance). Normalize the trajectory length to one control cycle, $\tau = 1$ in cycle units, and the equation becomes $C(x) = 1/\kappa(x)$. The tolerance has been absorbed into the choice of cycle rate; capacity is now purely a function of curvature.

**Step 3: Regularization.** The form $C = 1/\kappa$ has a problem at small curvature. As $\kappa \to 0$ (a flat region of the manifold, the regime where the linearization is most trustworthy), $C \to \infty$. But capacity is supposed to be bounded; it is a ratio that gates decisions, and a controller that interprets $C = \infty$ as "send infinite gain" will crash the vehicle. *We need a regularization.* The simplest regularization adds 1 to the denominator: $C = 1/(1+\kappa)$. Now at $\kappa = 0$, $C = 1$ (full confidence, full gain, no obstruction); at $\kappa \to \infty$, $C \to 0$ (no confidence, gain floor, full obstruction). The regularization is a design choice; it is not derived from the Duality. §2.4 below walks through why this specific regularization and not another.

**Step 4: Dimensional cleanup.** The $\kappa$ in step 3 still has units (the units of curvature, $1/\text{length}^2$). For the regularized expression $1/(1+\kappa)$ to make dimensional sense, $\kappa$ must be dimensionless—you cannot add 1 to a quantity that has units of $1/\text{length}^2$. The operational form replaces $\kappa$ with a dimensionless non-negative curvature obstruction $\widehat{\kappa}$ (*kappa-hat*; the hat indicates the dimensionless version), defined by multiplying the pointwise curvature by a length-scale squared:

$$\widehat{\kappa}(x) \;:=\; \ell^2 \cdot \|\Omega_x\|_g$$

where $\ell$ is the local control-cycle length scale ($\ell^2$ has units of $\text{length}^2$, canceling the $1/\text{length}^2$ units of $\|\Omega_x\|_g$). *Cancelación de unidades.* Unit cancellation. The hatted curvature obstruction is dimensionless and non-negative by construction, and the

operational form $C(x) = 1/(1 + \widehat{\kappa}(x))$ is now a well-defined dimensionless capacity in $(0, 1]$.

The four steps are: *shrink to a point, normalize the cycle, add the +1 regularizer, fix the units.* The reader who has stayed with me through these four moves now understands what the boxed equation in §1.3 was doing and what the operational specialization in Chapter 23 is doing. They are the same equation read at two scales.

§ 2.4

## *Why $1/(1 + \widehat{\kappa})$ and not another regularization*

A careful reader will object to step 3 of §2.3. The regularization is a design choice, not a derivation. The form $C = 1/(1 + \widehat{\kappa})$ is one member of a family of monotone-decreasing functions of $\widehat{\kappa}$ that all satisfy the bounded-capacity requirement. Why this one and not another?

The objection is fair. Three other candidates would also satisfy the basic requirements. I want to walk all four, name what each gets right and where each falls short, and explain why the framework picks the one it picks.

**Candidate A: $C = 1/(1 + \widehat{\kappa})$.** The framework's choice. Monotone-decreasing in $\widehat{\kappa}$. Bounded by 1 at $\widehat{\kappa} = 0$ and approaching 0 as $\widehat{\kappa} \to \infty$. Smooth (infinitely differentiable). Its derivative at $\widehat{\kappa} = 0$ is $-1$, which means small curvature produces a small linear drop in capacity—the same scaling as the Duality predicts.

**Candidate B: $C = e^{-\widehat{\kappa}}$.** Also monotone-decreasing, also smooth, also bounded by 1 at $\widehat{\kappa} = 0$. The exponential drops

faster than Candidate A: at moderate curvature ($\widehat{\kappa} = 1$), Candidate A gives $C = 0.5$, while Candidate B gives $C = 0.37$. The exponential is too aggressive in penalizing modest curvature; the framework wants graceful degradation, not panic. The exponential also has the wrong scaling in the small-curvature limit; its derivative at $\widehat{\kappa} = 0$ is $-1$ like Candidate A, but its second derivative is $+1$ rather than $+2$, so the curvature of the capacity-vs-obstruction curve disagrees with the Duality's prediction.

**Candidate C:** $C = 1/(1 + \widehat{\kappa}^2)$**.** Quadratic in $\widehat{\kappa}$ in the denominator. Monotone-decreasing, bounded by 1 at zero, approaching 0 at infinity. The problem is the small-curvature behavior: at $\widehat{\kappa} = 0$, the derivative is zero (a flat top), which means modest curvature produces *no* drop in capacity. The framework wants modest curvature to produce a modest drop, not no drop. Candidate C is too forgiving in the small-curvature regime.

**Candidate D:** $C = \tanh(\alpha/\widehat{\kappa})$**.** Hyperbolic tangent with a tuning parameter $\alpha$. This is what you get if you want a sharp threshold (a switch-like behavior). Candidate D is bounded, monotone, smooth, but it has an extra parameter $\alpha$ that must be tuned and that does not appear in the Duality. The framework prefers a parameter-free form, because a parameter-free form makes the universality claim across domains cleaner: no per-domain tuning means the equation transfers without modification.

The framework picks Candidate A because it is the unique member of the family that satisfies *all* of: smoothness, bounded by 1 at zero, monotone, approaching 0 at infinity, parameter-free, and matching the Duality's small-curvature derivative scaling. The other three candidates each fail at least one of these criteria. *No es la única función que cumple las cuatro primeras condiciones, pero sí es la única que cumple las seis.* It is not the only function that satisfies the first four conditions, but it is the only one that

satisfies all six.

The stability proof—the Lyapunov argument in Chapter 11 that establishes the gain floor's sizing condition—works with any member of the family that satisfies smoothness, boundedness, monotonicity, and a positive floor. The reader who prefers Candidate B or Candidate D can substitute in those forms, work through the stability proof, and arrive at a controller that flies. The flight controller does not depend critically on the choice of Candidate A. The framework's choice of A is a matching condition with the Duality's predicted scaling; it is the choice that minimizes the number of free parameters and maximizes the universality of the equation across domains. Both are aesthetic in a sense, but the aesthetic is in the service of the framework's load-bearing claim that the same equation runs across all of them.

§ 2.5

## *The signed Ricci scalar $R$ versus the non-negative obstruction $\widehat{\kappa}$*

A reader who has studied Riemannian geometry will, by this point in the chapter, be raising an objection. The symbol $\kappa$ in Riemannian geometry usually denotes *scalar curvature*, which is the contraction of the Ricci tensor and which is a *signed* quantity. Scalar curvature can be positive (spheres), zero (Euclidean space), or negative (hyperbolic surfaces, saddles, the Gaussian Fisher manifold). The framework's operational equation $C = 1/(1 + \widehat{\kappa})$ assumes $\widehat{\kappa} \geq 0$. If $\widehat{\kappa}$ were the signed Ricci scalar, then at $\widehat{\kappa} = -1$ the denominator would vanish, $C$ would diverge, and for $\widehat{\kappa} < -1$ the capacity would go *negative*, which makes no operational

sense.

The objection is correct, and the framework's response is to distinguish two different quantities with two different symbols.

**The signed Ricci scalar $R$.** The standard Riemannian-geometric scalar curvature, the contraction of the Ricci tensor over an orthonormal frame. $R$ is signed; it can take any real value. $R$ is what appears in Einstein's field equation of general relativity ($G_{\mu\nu} + \Lambda g_{\mu\nu} = (8\pi G/c^4)T_{\mu\nu}$, where $G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2}R\, g_{\mu\nu}$). $R$ is what the Riemann tensor reduces to when you take the simplest scalar invariant.

**The non-negative obstruction $\widehat{\kappa}$.** The framework's operational quantity. Dimensionless by construction (multiplied by $\ell^2$). Non-negative by construction (it is a norm of the curvature 2-form, not a contraction). Equal to zero only when the manifold is locally flat; positive everywhere else.

The two quantities are related but not equal. In a region where $R > 0$ (positive sectional curvature, like a sphere), the norm $\|\Omega\|$ is positive, so $\widehat{\kappa}$ is positive. In a region where $R = 0$ (flat space), the norm $\|\Omega\|$ is zero, so $\widehat{\kappa}$ is zero. In a region where $R < 0$ (negative sectional curvature, hyperbolic), the norm $\|\Omega\|$ is *still positive*, because a norm is non-negative regardless of the sign of what it is the norm of—so $\widehat{\kappa}$ is positive in negatively-curved regions too. Both positively-curved and negatively-curved regions impose a curvature tax; only flat space is free.

This is the key observation that licenses the framework's use of $\widehat{\kappa}$ instead of $R$. The Field Equation says capacity is bounded by curvature; the cost of curvature is the same whether you are on a sphere or in a saddle. Both shapes are obstructions to flat approximation. $\widehat{\kappa}$ captures this by taking the magnitude rather than the sign. $R$ does not, because $R$ was defined for a different question (Einstein's question, about how matter sources spacetime curvature, where the sign matters because matter

sources positive curvature and dark energy sources effectively negative curvature). Einstein's question is not the framework's question.

The Gaussian Fisher manifold is the cleanest case for showing why this distinction matters. The Fisher information metric on the space of multivariate Gaussian distributions makes the parameter space into a Riemannian manifold whose scalar curvature $R$ is *constant and negative*, by a classical calculation due to Skovgaard (1984; see also Amari and Nagaoka 2000, Chapter 7, for the modern treatment).[1] If the framework's confidence formula used the signed Ricci scalar $R$, then on the Gaussian Fisher manifold the confidence denominator $1+R$ would be less than one for large enough Gaussian dimension and would eventually go negative, breaking the formula. The framework's confidence formula uses $\widehat{\kappa}$, the non-negative norm, which is positive on the Gaussian Fisher manifold (and on every other manifold with non-trivial curvature, regardless of sign); the confidence is bounded and well-defined; the Kalman filter chapter (Chapter 10) can use the formula directly.

The distinction between $R$ and $\widehat{\kappa}$ is the kind of detail a reviewer will press on. The framework's response is in this section; the deeper version of the response is in §23.3 of Chapter 23, where the operational form is derived from the engineering side and the dimensional analysis is walked in full.

[1]The specific numerical value depends on convention—whether the parameterization uses the mean alone, the mean and covariance together, or a different reparameterization; whether the Fisher metric is normalized to unit determinant; whether scalar curvature is the trace of Ricci or a different contraction. These conventions vary across the statistical and geometric literature. The fact that does not vary, and that this section depends on, is the *sign*: scalar curvature of the Gaussian Fisher manifold is negative under any standard convention.

§ 2.6

## *What this chapter has set up; what Chapter 3 walks*

This chapter has done six things.

It has placed you in front of a laptop on the night the mass gap became visible as a hole in a spectrum, and named the moment as the genesis of the Field Equation that drives the rest of the book. It has separated the three letters of $C = \tau/K$ into the engineer's tolerance, the world's curvature, and the resulting capacity, and explained why an equation with two letters could not have done the work. It has stated the full trajectory-integrated form of the Field Equation and grounded it in the ruler-and-road picture from §1.3. It has walked the four-step motivation from the foundational identity to the operational form $C = 1/(1 + \widehat{\kappa})$ that ICARUS uses at flight time. It has compared the framework's chosen regularization against three alternatives and explained why the chosen form is the one with the cleanest match to the Duality and the fewest free parameters. And it has distinguished the dimensionless non-negative obstruction $\widehat{\kappa}$ from the signed Ricci scalar $R$ that a Riemannian geometer would default to, with the Gaussian Fisher manifold as the case that shows why the distinction is operational rather than aesthetic.

Chapter 3 lays down the mathematical preliminaries that the rest of the book uses freely. Fiber bundles in formal detail; sections and parallel transport; the Ehresmann connection structure that Chapter 1 introduced via the bra; holonomy and the Ambrose–Singer theorem; the quaternion double cover $\mathbb{S}^3 \to \mathrm{SO}(3)$ in the level of detail that Chapter 23's claim 2 (unit-quaternion invariant enforcement) requires. Chapter 3 is the chapter that the differential geometer reads quickly and the

lay reader reads slowly, and it is written to honor both.

Then Part II picks up the F-16 longitudinal short-period model, and the worked-example arc of the book begins.

*Vamos.*

Chapter 3

# *Mathematical preliminaries for the chapters that follow*

*Mathematics is the art of giving the same name*
*to different things.*
*Henri Poincaré, Science and Method (1908)*

§ 3.0

## *Where the metaphor becomes a theorem*

This is the chapter I almost cut.

A preliminaries chapter is, by definition, a chapter nobody reads for pleasure. The reader who needs it skips it; the reader who doesn't already knows. There is a real argument—an argument I have heard from editors and friends and the version of myself that wanted this book to feel like a single uninterrupted breath—for moving this material into an appendix and getting on with the airplanes.

I kept it for two reasons.

First: the rest of the book does load-bearing work with the terms I introduced casually in §1.2. I claim, in Chapter 5, that

a quadcopter is two coupled fiber bundles. I claim, in Chapter 6, that the F-16 longitudinal model has gauge symmetries. I claim, in Chapter 20, that ICARUS computes a flight controller as a section of a controller bundle. If you are going to take any of those claims seriously, the definitions need to be sitting somewhere you can reach without leaving the book. I would rather you encounter them in a chapter where my voice is still consistent than in a back-of-the-book reference that reads like everyone else's reference.

Second—and this is the one that decided it—there is exactly one theorem in this chapter that is doing more work than any other theorem in the book. It is called Ambrose–Singer. It was published in 1953. It says, in language a differential geometer will accept, what I said in §1.2 with a bra: *the local check propagates because curvature is the local source of holonomy, and when curvature is zero the holonomy is trivial.*

I had been carrying the informal version of this claim for years before I learned the formal version. I encountered Ambrose–Singer in graduate school and did not see the connection. The theorem sat in my notes; the bra-strap intuition sat in my body; they did not speak to each other. The morning they finally did—band-straight, straps-straight, the recognition—I was standing in my bathroom in front of a mirror and I said out loud, to nobody, *esto es Ambrose–Singer.* This is Ambrose–Singer. Whatever I had been doing for years in front of mirrors had been doing the theorem.

That is the moment that earned this chapter its place in the book.

I will not prove Ambrose–Singer. The proof is in Kobayashi–Nomizu Volume I, Chapter II, and it is not a proof I am going to improve on. I will state it carefully enough that you can recognize it as the same claim I made in §1.2. I will state the definitions it rests on. I will spend the rest of the chapter on the one

piece of topology—$\pi_1(\mathrm{SO}(3)) \cong \mathbb{Z}/2$—that I keep promising the quaternion lift handles, so that by the time we get to Chapter 6 and start writing down the F-16's state space you know exactly what topological obstruction the controller is dodging and why.

Read this chapter once. Read §3.3 twice. Skim the rest. Come back when a later chapter cites it.

The next chapter is about birds.

§ 3.1

## *Fiber bundles, formally*

Before the formal definition: remember what the kid in §1.0 did when she looked at the lawnmower-bike. She saw the lawnmower's blades, spinning, and the BMX's gears, also spinning. She recognized that both of those things were the same kind of object—rotational motion around an axis—living on the same circle $S^1$. The blades were one fiber over that circle (angular position of the blade); the gears were another fiber (angular position of the gear); the bike's geometry was a third (where each rotating part was bolted). The kid recognized that all of these were sections of fiber bundles over the same base manifold ($S^1$ for the rotations, $\mathbb{R}^3$ for the bike's spatial layout), without ever using that vocabulary. The recognition was structural: *these different-looking things are the same kind of thing.* This section gives the formal version of what the kid was doing.

A **fiber bundle** is a quadruple $(E, M, \pi, F)$ where:

- $E$ is a smooth manifold called the *total space*;
- $M$ is a smooth manifold called the *base*;

- $F$ is a smooth manifold called the *typical fiber*;
- $\pi : E \to M$ is a smooth surjection;

subject to the requirement of *local triviality*: for every $p \in M$ there is an open neighborhood $U \ni p$ and a diffeomorphism

$$\varphi : \pi^{-1}(U) \longrightarrow U \times F$$

such that $\mathrm{pr}_1 \circ \varphi = \pi|_{\pi^{-1}(U)}$, where $\mathrm{pr}_1 : U \times F \to U$ is projection onto the first factor.

The condition $\mathrm{pr}_1 \circ \varphi = \pi$ says: $\varphi$ respects the projection. It is allowed to scramble the fiber, but it sends each fiber $\pi^{-1}(q)$ into $\{q\} \times F$.

Local triviality is what makes the bundle look, in a small enough neighborhood, like an ordinary product $U \times F$. Globally, the bundle may be twisted. The simplest twisted bundle is the Möbius band, which is locally (arc) $\times [-1, 1]$ but globally not $S^1 \times [-1, 1]$. The simplest non-trivial bundle most readers encounter without noticing is the tangent bundle of $S^2$: locally it looks like (patch of sphere) $\times \mathbb{R}^2$, but globally it is non-trivial—a fact known to anyone who has ever tried to comb a sphere's hair and felt the cowlick.

**The bra example, made formal.** In §1.2 I described a bra as a fiber bundle with base $M =$ the band and fiber $F =$ the strap. Let me actually write down what that means.

- $M = S^1$ (the band, topologically a circle).
- $F = [0, \ell]$ (the strap, an interval of length $\ell$ running orthogonally to the band).
- $E =$ the bra in its entirety, a 2-manifold with boundary.
- $\pi : E \to M$ sends each point of the bra to the nearest point on the band.

On any small open arc $U \subset S^1$, the bra restricted to $U$ is diffeomorphic to $U \times [0, \ell]$. The two trivializations on overlapping arcs agree via the identity (the strap configuration does not twist as you go around the band).

This is what makes the bra a *trivial* bundle: globally, the bra is diffeomorphic to $S^1 \times [0, \ell]$. The band-straight check at one point of $S^1$ propagates to the whole bundle because there is no twist for it to fail across. If the bra were a Möbius bra—if the strap geometry flipped sign as you went once around the band—the local check would not propagate. The bra-strap principle of §1.2 was implicitly using triviality. Ambrose–Singer (§3.3) will tell us what to do when the bundle is not trivial.

**The tangent bundle.** Let $M$ be a smooth manifold. The *tangent bundle* is

$$TM = \bigsqcup_{p \in M} T_p M$$

with $\pi : TM \to M$ sending each tangent vector to its basepoint and typical fiber $F = \mathbb{R}^n$ (where $n = \dim M$). Local trivializations come from coordinate charts: in a chart $(U, x^1, \ldots, x^n)$, every tangent vector at $p \in U$ is uniquely $v^i \partial/\partial x^i|_p$, and the map $v \mapsto (p, v^1, \ldots, v^n)$ is the local trivialization $\pi^{-1}(U) \to U \times \mathbb{R}^n$.

The change-of-coordinates transformations between trivializations are the Jacobian matrices of the coordinate maps. They take values in $\mathrm{GL}(n, \mathbb{R})$. The bundle is trivial (globally a product) only for certain manifolds—the *parallelizable* ones. The parallelizable spheres, among the positive-dimensional cases, are $S^1$, $S^3$, and $S^7$—a list whose finiteness is one of the more arresting facts in topology, and one that ties directly to the existence of $\mathbb{R}$, $\mathbb{C}$, $\mathbb{H}$, and the octonions $\mathbb{O}$ as the four normed division algebras over the reals. Every Lie group is parallelizable, which is why $\mathrm{SO}(3)$, considered as a 3-manifold, has trivial tangent bundle even though—as we will see in §3.4—it is non-trivial in a different sense.

**Principal bundles.** A *principal $G$-bundle* is a fiber bundle whose fiber is a Lie group $G$ and which carries a smooth right action of $G$ on the total space, free and transitive on each fiber. Principal bundles are the natural setting for gauge theory: $G$ is the gauge group, and the principal bundle records how the gauge frame varies over the base.

For our purposes the most important principal bundle is the *frame bundle* of a manifold $M$: the bundle whose fiber over $p \in M$ is the set of ordered bases of $T_pM$. The structure group is $\mathrm{GL}(n, \mathbb{R})$. For a Riemannian manifold you can restrict to orthonormal frames, giving the *orthonormal frame bundle* with structure group $\mathrm{O}(n)$.

For a quadcopter, the natural principal bundle has base $M = \mathbb{R}^3 \times \mathbb{R}$ (position and time) and structure group $G = \mathrm{SO}(3)$ (the rotation group, encoding the vehicle's body-frame attitude). The flight controller is, formally, a section of an associated vector bundle—the controller bundle—built from this principal bundle. We will not need that level of formality until Chapter 20. But it will be sitting there waiting.

§ 3.2

## *Sections, connections, parallel transport*

A **section** of a fiber bundle $\pi : E \to M$ is a smooth map $s : M \to E$ such that $\pi \circ s = \mathrm{id}_M$. Equivalently: $s$ picks one point in each fiber, smoothly.

For the bra: a section assigns one position-along-the-strap to each point of the band. "The top of the strap" is a section. "The bottom of the strap" is another. "The middle of the strap" is a third.

For the tangent bundle: a section is a *vector field*. Every vector field on $M$ is exactly a section of $TM$. The Hairy Ball Theorem (Brouwer, 1912) says: $S^2$ admits no nowhere-vanishing section of its tangent bundle. There is no smooth choice of nonzero tangent vector at every point of the sphere. This is what makes $TS^2$ a non-trivial bundle.

For the quadcopter's configuration bundle: a section assigns one attitude $q \in \mathrm{SO}(3)$, and its time-derivative, to each instant. A trajectory of the quadcopter is exactly a section over the time interval of the maneuver.

**Connections.** This is the part where the formalism earns its keep.

A bundle by itself does not let you compare points in different fibers. Two strap-positions at two different points of the band are, formally, elements of two different copies of $[0, \ell]$. The bundle structure alone gives no way to say "this point of one fiber corresponds to that point of the other."

A **connection** is exactly this missing structure. It is a choice of how to identify infinitesimally nearby fibers.

The precise definition. Let $\pi : P \to M$ be a principal $G$-bundle. A *connection* on $P$ is a smooth $\mathfrak{g}$-valued 1-form $\omega$ on $P$, where $\mathfrak{g}$ is the Lie algebra of $G$, satisfying:

1. *$G$-equivariance:* $R_g^* \omega = \mathrm{Ad}(g^{-1})\omega$ for all $g \in G$, where $R_g$ is the right action of $g$ on $P$ and Ad is the adjoint representation.

2. *Vertical reproduction:* for every $X \in \mathfrak{g}$, if $\tilde{X}$ is the vertical vector field on $P$ generated by $X$, then $\omega(\tilde{X}) = X$.

Equivalently—and more useful intuitively—a connection is a smooth choice of *horizontal subspace* at each point of $P$. At every $e \in P$, the tangent space $T_e P$ splits canonically into a vertical subspace (tangent to the fiber) plus a horizontal complement

determined by $\omega$:

$$T_eP \;=\; V_e \oplus H_e$$

The horizontal subspace is the kernel of $\omega$ at $e$.

This is what a connection *is*, mechanically: a way of saying "these tangent directions stay in the fiber" (vertical) versus "these tangent directions move you to nearby fibers in a way that respects the gauge" (horizontal).

**Parallel transport.** Given a connection and a smooth curve $\gamma : [0,1] \to M$ starting at $p = \gamma(0)$, and given a starting fiber element $e_0 \in \pi^{-1}(p)$, there is a unique smooth curve $\tilde{\gamma} : [0,1] \to P$ such that:

- $\pi \circ \tilde{\gamma} = \gamma$ (the curve in $P$ lifts the curve in $M$);
- $\tilde{\gamma}(0) = e_0$ (it starts at the prescribed fiber element);
- $\dot{\tilde{\gamma}}(t) \in H_{\tilde{\gamma}(t)}$ for all $t$ (the tangent vector to $\tilde{\gamma}$ at each point is horizontal).

The endpoint $P_\gamma(e_0) := \tilde{\gamma}(1) \in \pi^{-1}(\gamma(1))$ is the *parallel transport* of $e_0$ along $\gamma$. Existence and uniqueness of the horizontal lift follow from the ODE theory: the horizontality condition is a first-order ODE on $P$, and the connection makes its coefficients smooth.

This is what "the bra strap propagates the band's tightness" means formally: parallel transport along the band carries the strap configuration from one point of the base to another. The local check at one point of the band determines, via parallel transport, the strap configuration at every other point on the same connected piece of the bundle.

**Curvature.** The *curvature* of a connection $\omega$ is the 2-form

$$\Omega \;=\; d\omega + \tfrac{1}{2}[\omega, \omega]$$

also $\mathfrak{g}$-valued. The bracket $[\omega, \omega]$ is the wedge product of $\omega$ with itself in the Lie algebra (it is nonzero because $\omega$ is matrix-valued, not scalar-valued).

Two equivalent characterizations of curvature, both load-bearing for the rest of the book:

- *Curvature is the obstruction to integrability of the horizontal distribution.* If $\Omega = 0$, the horizontal distribution $H$ is integrable: through every point of $P$ there passes a maximal horizontal submanifold. If $\Omega \neq 0$, horizontal vector fields fail to close under the Lie bracket, and you cannot find such submanifolds.

- *Curvature is the infinitesimal failure of parallel transport to be path-independent.* For any infinitesimal loop in the base—a parallelogram spanned by two tangent vectors $X, Y$ at $p$—parallel transport around the loop differs from the identity by $\Omega_p(X, Y)$ to first order in the loop area.

This is the precise sense in which curvature is "local non-flatness." It is also why I have insisted, since §1.3, that curvature is the right currency for measuring how badly the local check can fail to propagate.

## § 3.3

## *Holonomy and the Ambrose–Singer theorem*

Now we collect what §3.2 set up.

Fix a principal $G$-bundle $\pi : P \to M$ with connection $\omega$ and curvature $\Omega$. Fix a base point $p \in M$ and a fiber element $e \in \pi^{-1}(p)$.

Consider every *piecewise-smooth loop* at $p$: a closed curve $\gamma : [0,1] \to M$ with $\gamma(0) = \gamma(1) = p$, smooth except possibly at finitely many points. Parallel transport around any such loop gives a map

$$P_\gamma : \pi^{-1}(p) \longrightarrow \pi^{-1}(p)$$

from the fiber over $p$ to itself. Because parallel transport along a concatenation of loops is the composition of the parallel transports, and because the constant loop transports trivially, the set of all such maps forms a group under composition.

**Definition.** The *holonomy group* at $p$ (with respect to the connection $\omega$) is

$$\mathrm{Hol}_p(\omega) \;=\; \{P_\gamma : \gamma \text{ a piecewise-smooth loop at } p\} \;\subseteq\; G.$$

The holonomy group is what records how much the connection "twists" as you transport around loops. It is the formal object that measures the failure of the local check to propagate globally.

A few facts that orient the definition:

- If $\Omega \equiv 0$ everywhere (the connection is *flat*) and $M$ is simply connected, then $\mathrm{Hol}_p = \{e\}$. Trivial holonomy. The local check propagates exactly.
- If $\Omega \equiv 0$ but $M$ is not simply connected, parallel transport depends only on the homotopy class of the loop, so $\mathrm{Hol}_p$ is the image of a representation $\rho : \pi_1(M,p) \to G$. The image may be a proper subgroup of $G$ and the representation may have a kernel; the holonomy group is not in general isomorphic to $\pi_1(M,p)$, only a quotient of it. For a flat connection on a torus with non-trivial monodromy along both loops, $\mathrm{Hol}_p$ is the subgroup of $G$ generated by the two monodromy elements.
- If $\Omega \not\equiv 0$, the holonomy can be much bigger than $\pi_1$. Cur-

vature is generating new holonomy that the topology of $M$ alone could not see.

The Ambrose–Singer theorem makes this third bullet quantitative.

**Theorem (Ambrose–Singer, 1953).** *Let $\pi : P \to M$ be a principal $G$-bundle with connection $\omega$ and curvature $\Omega$. Fix a point $e \in P$ and let $P(e)$ denote the set of all points in $P$ reachable from $e$ by horizontal piecewise-smooth curves. Then the Lie algebra of the holonomy group at $\pi(e)$ is*

$$\mathfrak{hol}_e \;=\; \operatorname{span}_{\mathbb{R}}\Big\{\, \Omega_{e'}(X,Y) \;:\; e' \in P(e),\; X, Y \in H_{e'} \,\Big\}.$$

In English: the Lie algebra of the holonomy group is generated by the curvature 2-form, evaluated on horizontal vectors, at every point reachable from $e$ along horizontal paths.

Curvature is the local source of holonomy. Holonomy is the global accumulation of curvature.

The bra-strap principle, restated formally one last time:

> *If the curvature $\Omega$ vanishes on a neighborhood $U \subseteq M$, then for every loop $\gamma \subset U$ contractible in $U$, the holonomy $P_\gamma$ is the identity. The local check at one point of $U$ propagates without modification to every other point of $U$ reachable along contractible loops.*

That is the theorem. That is the morning. *Esto es Ambrose–Singer.*

**Why this theorem carries the rest of the book.** Four places, listed in order of appearance:

1. *Control (Chapters 6–9).* Curvature controls how badly nonlinear effects spoil linear feedback predictions. If the linearization of the plant is exactly flat in a neighborhood

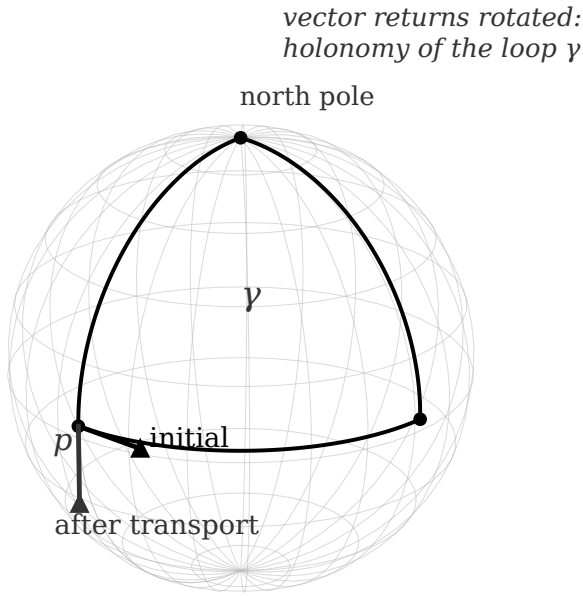



Figure 3.1: Holonomy on the sphere. Parallel-transport a tangent vector around the great-circle triangle $\gamma$ at base point $p$. The vector rotates as it travels each side of the triangle; when it returns to $p$ it has acquired a net rotation equal to the spherical excess (the triangle's enclosed area on the unit sphere). The accumulated rotation is the *holonomy* of $\gamma$. Ambrose–Singer makes this quantitative: the curvature 2-form at each interior point contributes the local rotation rate, and the holonomy is the integral over the enclosed region.

of the equilibrium, gain-scheduled linear control works without modification on that neighborhood. If curvature is present, the gain schedule has to track it. This is the seed of the Davis Field Equation $C = \tau/K$ from Chapter 2.

2. *Estimation (Chapters 10–16).* Parallel transport along a state-trajectory is what propagates a Kalman filter's covariance matrix from time $t$ to time $t + dt$. The Riccati equation is the curvature term that makes this propagation non-trivial. The Fisher manifold of a probabilistic model has curvature, and that curvature is precisely what controls how badly the linear-Gaussian approximation fails.

3. *Adaptation (Chapters 12–13).* Adaptive controllers move on a parameter manifold $\Theta$. The natural metric on $\Theta$ is the Fisher information metric; the curvature of that metric is the obstruction to a single set of gains working across the whole parameter range. The Curvature Floor conjecture in Chapter 13 is, structurally, an Ambrose–Singer statement on $\Theta$.

4. *ICARUS (Chapter 20).* Holonomy-gated control is the architectural commitment that the controller's effective gain is modulated by an estimate of local holonomy along the predicted trajectory. The estimate is computed from the curvature obstruction $\hat{\kappa}$. The modulation schedule $m(C)$ is exactly the operational consequence of Ambrose–Singer: "curvature is the local source of holonomy, so gate the controller on a local-curvature query rather than on a global-holonomy computation we can't afford to do in real time."

If you understand the statement of Ambrose–Singer—not the proof, just the statement, just the words "curvature is the local

source of holonomy"—you understand why this whole book holds together.

You can put the chapter down for a minute. The next section is topology.

§ 3.4

## *Quaternions, $S^3$, and $\pi_1(\mathrm{SO}(3)) \cong \mathbb{Z}/2$*

SO(3) is the group of rotations of three-dimensional space. As a manifold it is three-dimensional, compact, and connected. It is also a Lie group, which means its tangent bundle is trivial. By the criteria of §3.1, you might expect SO(3) to be "nice."

It is not.

SO(3) is not simply connected. There exist loops in SO(3) that cannot be contracted to a point. The space of those non-contractible loops, modulo homotopy, has order exactly two:

$$\pi_1(\mathrm{SO}(3)) \;\cong\; \mathbb{Z}/2.$$

This is the single topological fact that makes flight control on SO(3) non-trivial.

**The Dirac belt trick.** The clearest demonstration is the trick Paul Dirac used to lecture on. Hold one end of a belt fixed in your left hand. With your right hand, rotate the other end of the belt through 360° about a vertical axis. The belt now has a twist; the twist cannot be removed by translating the right hand around without further rotation. The twist is the homotopy class of the loop you just drew in SO(3).

Now rotate the right hand through another 360° in the same direction. The belt now has a 720° twist—and that twist *can* be

removed by passing the right hand over or under the belt, without further rotation, returning the belt to its starting configuration with both hands held fixed.

A 360° rotation is a loop in SO(3) that is not contractible. A 720° rotation is a loop that is. Loops in SO(3) come in exactly two homotopy classes. That is what $\pi_1(\mathrm{SO}(3)) \cong \mathbb{Z}/2$ says.

**The quaternion algebra.** The quaternions $\mathbb{H}$ are a four-dimensional real algebra with basis $\{1, i, j, k\}$ and multiplication

$$i^2 = j^2 = k^2 = ijk = -1,$$

which forces $ij = k$, $ji = -k$, and the cyclic permutations. Multiplication is associative but not commutative; every nonzero quaternion has a multiplicative inverse. A quaternion can be written $q = q_0 + q_1 i + q_2 j + q_3 k$; its *conjugate* is $q^* = q_0 - q_1 i - q_2 j - q_3 k$; its *norm* is $|q|^2 = qq^* = q_0^2 + q_1^2 + q_2^2 + q_3^2$.

The set of unit quaternions

$$S^3 \;=\; \{q \in \mathbb{H} : |q| = 1\}$$

is a smooth submanifold of $\mathbb{H} = \mathbb{R}^4$, topologically the 3-sphere, algebraically a Lie group under quaternion multiplication.

**The covering map.** For each $q \in S^3$, the map

$$\rho(q) : v \;\longmapsto\; q\, v\, q^*$$

sends pure imaginary quaternions to pure imaginary quaternions. Identifying pure imaginary quaternions with $\mathbb{R}^3$ via $v_1 i + v_2 j + v_3 k \leftrightarrow (v_1, v_2, v_3)$, the map $\rho(q) : \mathbb{R}^3 \to \mathbb{R}^3$ is a rotation. The assignment $q \mapsto \rho(q)$ is a Lie group homomorphism

$$\rho : S^3 \;\longrightarrow\; \mathrm{SO}(3).$$

Three facts about $\rho$:

- It is surjective. Every rotation of $\mathbb{R}^3$ comes from some unit quaternion.
- Its kernel is exactly $\{+1, -1\} \cong \mathbb{Z}/2$. The two quaternions $q$ and $-q$ produce the same rotation: $(-q)v(-q)^* = qvq^*$.
- It is a smooth covering map. Locally, $\rho$ is a diffeomorphism. Globally, it is two-to-one.

So we have a short exact sequence

$$1 \;\to\; \mathbb{Z}/2 \;\to\; S^3 \;\xrightarrow{\rho}\; \mathrm{SO}(3) \;\to\; 1.$$

$S^3$ is the *double cover* of SO(3). And because $\pi_1(S^3) = 0$ (the 3-sphere is simply connected) and $\rho$ is a covering map, $S^3$ is in fact the *universal cover* of SO(3). The covering deck group is $\mathbb{Z}/2$. That is the topological explanation for the belt trick: lifting a 360° loop in SO(3) to $S^3$ gives a path from $q$ to $-q$, which is not a loop; lifting a 720° loop gives a path from $q$ back to $q$, which is.

**Why this matters for flight control.** A quadcopter's attitude lives in SO(3). The controller has to track it, command changes to it, and compute attitude errors against a reference. The question is: what coordinates does the controller use?

If you parameterize SO(3) with three numbers—Euler angles, axis-angle, anything—you run into singularities. Euler angles have gimbal lock at ±90° pitch. Axis-angle has a singularity at the identity rotation (the axis is undefined). These are not numerical issues. They are forced by the topology of SO(3): a three-manifold with $\pi_1 = \mathbb{Z}/2$ cannot be smoothly parameterized by a single chart on $\mathbb{R}^3$. There is always a hole somewhere.

The standard fix is the lift. Track the attitude on $S^3$ instead of SO(3). The 3-sphere $S^3$ is smoothly embedded in $\mathbb{R}^4$ by the global redundant parameterization $(q_0, q_1, q_2, q_3)$ with one constraint

$|q|^2 = 1$. The embedding is not a chart in the strict sense (a chart would map an open set diffeomorphically to $\mathbb{R}^3$), but it does avoid the singularities of any three-coordinate parameterization: the four-component representation never collapses two distinct rotations to the same point in coordinates. The price you pay is that the lift is two-to-one: each attitude in SO(3) corresponds to two quaternions in $S^3$, namely $q$ and $-q$.

You pay that price by picking a sign convention. The *short-arc convention* says: prefer the $q$ with $q_0 \geq 0$, equivalently the rotation that minimizes the rotation angle from the reference. This is what gets used in practice. Every quaternion-attitude controller you have ever seen on an autopilot enforces, somewhere, a sign correction that picks the short arc.

Without short-arc selection, the cost is direct and measurable. The standard quaternion attitude-error formulation is multiplicative, $q_{\text{err}} = q_{\text{cmd}}^{-1} \otimes q_{\text{actual}}$, with the vector part of $q_{\text{err}}$ used as the error signal. Even in this formulation, $q_{\text{err}}$ and $-q_{\text{err}}$ represent the same physical attitude error, and the controller has to pick a sign convention to choose between them. A naive implementation—computing $q_{\text{error}} = q_{\text{cmd}} - q_{\text{actual}}$ as a Euclidean subtraction in $\mathbb{R}^4$, or using the multiplicative form without sign correction—hits a sign flip the moment the trajectory passes near $\theta = \pi$ and commands a control effort in the wrong direction. The fixed-gain quaternion LQR baseline of Chapter 20 crashes 100% of trials on multirotor flip exactly because of this. The corrected baseline—multiplicative error with short-arc sign selection—flies the maneuver but at a high control-effort cost. Both ICARUS and the corrected baseline are paying for the topology of SO(3), just in different currencies.

This is not a tuning issue. It is a topological fact about SO(3) propagating through the controller, and no choice of gains alone repairs it; only the right error formulation does.

I will develop this carefully in Chapter 14. For now: every time

you see a quaternion-attitude controller in this book, remember that the lift is doing topological work.

§ 3.5

## *Why this preliminaries chapter exists*

I told you at the start of this chapter why I kept it. Let me end by telling you what I want you to take from it.

There are two pieces of mathematics in this chapter that you cannot do this book without.

The first is the statement of Ambrose–Singer. Not the proof. Not the technical hypotheses. The statement, in the form: *curvature is the local source of holonomy.* If a later chapter cites Ambrose–Singer and you can recall that sentence, you have what you need. The bra of §1.2, the ruler of §1.3, the field equation of Chapter 2, the holonomy-gated control of Chapter 20—all of them are operational consequences of that sentence.

The second is the topological fact $\pi_1(\mathrm{SO}(3)) \cong \mathbb{Z}/2$. Not the homotopy theory. Not the universal cover construction. The fact, in the form: *rotations cannot be coordinatized smoothly without choosing a double cover.* If a later chapter talks about the quaternion lift, the short-arc convention, or the cost of crossing $\theta = \pi$, the explanation is always traceable to this fact.

The rest of the chapter—the formal bundle definition, the connection 1-form, the curvature 2-form, the Lie-algebra hypotheses on Ambrose–Singer, the construction of the quaternion algebra from first principles—is here because I would rather have it sitting in the book than make you go find it. Use it as a reference. The book will not assume you have it memorized.

What comes next is Part II.

Part II is about the plant. Three chapters. Chapter 4 reads aerodynamic flight as a sameness statement: *birds and airplanes are doing the same math*, both negotiating between an attitude bundle and an aerodynamic-force bundle, with curvature where the negotiation is hardest. Chapter 5 writes a quadcopter as two coupled fiber bundles and shows what control means as a section over the coupled base. Chapter 6 promotes the F-16 longitudinal model from the `fal-core` working notes into a chapter, with the gauge symmetries spelled out explicitly.

By the end of Part II, you should be able to read a state-space model the way I do: as a coordinate-dependent presentation of a section of a controller bundle, with a gauge group that is much larger than the equation makes it look. Then Part III—frequency, optimal, robust—will take that geometric reading and run it through the canonical synthesis machinery, Bode and LQR and $H_\infty$, and show what each of those tools is computing when you stop pretending the manifold is flat.

The preliminaries are over. The airplanes start now.

*Vamos.*

# *Lab I: Build a bundle, write a section, read it back.*

*Time: about 30 minutes. Materials: Python 3.10 or later, a text editor, and 20 minutes of patience. Optional: a Jupyter notebook, if you prefer that workflow to a REPL.*

Chapter 3 formalized the vocabulary: fiber bundles, sections, connections, parallel transport, holonomy. This lab turns the vocabulary into something you can hold in your hands. You are going to build a fiber bundle by typing it out, write it down by hand in a serialization format called DHOOM, and parse it back into Python with a library that knows how to read it as a bundle. By the end you will have constructed a fiber bundle, written a section of it, modified the section by introducing a deviation from the zero section, and watched the framework's compression principle do what fiber-bundle geometry says it should do.

**What DHOOM is.** DHOOM stands for *Davis Human-readable Optimized Object Markup*. It is a serialization format I developed alongside ICARUS and Marcella, applying the same fiber-bundle geometry to structured data that the rest of this book applies to flight control and language modeling. The library's documentation puts the design principle this way: *what happens when a differential geometer looks at JSON and says, "you're transmitting the fiber with every section."* DHOOM encodes the same data model as JSON—objects, arrays, strings, numbers, booleans, null—but quotients out the structural

redundancy that JSON imposes by repeating field names on every record and listing constant fields explicitly every time they recur. A typical DHOOM file is forty to sixty percent smaller than the JSON it encodes, with no information loss. The format is documented in U.S. Patent Application No. 64/008,940 (filed March 18, 2026) and has reference implementations in Python, TypeScript, Rust, Go, .NET, and Java. For this lab we use the Python library.

DHOOM is part of the framework's *substrate stack*, not a parallel deployment. The stack has three pieces: Gigi—the fiber-bundle database engine (U.S. Provisional App. No. 64/045,889, the sibling filing to the ICARUS architecture; documented in Chapter 18)—is the substrate that stores and queries bundles at flight speed. GQL is the query language against Gigi; the `SPECTRAL` and `INVARIANT` queries that Chapter 7's sidebar referenced are written in GQL. DHOOM is the wire format the substrate uses to move bundles in and out: when ICARUS reads its state-history bundle from Gigi, it reads DHOOM; when Marcella reads its token-embedding bundle from Gigi, it reads DHOOM. The framework's engineering deployments—ICARUS (Chapter 20, flight control), Marcella (Chapter 21, transformer replacement), and Chihiro (plasma confinement, provisional only)—all run on top of this stack. DHOOM is the piece of the stack you can install today, in twenty minutes, with `pip install dhoom`. Gigi, GQL, and the three deployments are reported in their own chapters.

The dataset is small and the code is short. The point is not the code. The point is the recognition: when you write a DHOOM file, you are writing geometry. The format makes the bundle structure explicit, and the parser reads sections off the file the way differential geometry reads sections off a bundle.

**One note about $\widehat{\kappa}$ across the labs.** Chapter 10 §10.4 names the framework's design-choice $\widehat{\kappa}$ as a *family* of monotone non-

negative scalar functions of the joint covariance's conditioning. Different labs pick the variant most natural for their domain: Lab II uses a normalized squared-state form for the LQR closed loop; Lab III uses a windowed regressor autocorrelation for PE monitoring; Lab IV uses the log-determinant of the joint covariance to make the cross-correlation block's contribution visible; Lab V uses a baseline-normalized trace; Lab VI uses a barrier-overlap integral along the cost-manifold reference. All are valid design-choice $\widehat{\kappa}$ in the framework's sense—monotone, non-negative, dimensionless. The reader who notices the variant changes across labs is reading correctly. The family is the invariant; the specific function is the design choice for the domain.

**One framework note before we start.** This is Lab I. There is no Lab 0—Roman numerals begin at I, just as the geometric naturals begin at $\mathbb{G}_1$. The seed is the singleton: one element, no connections, formally present but pre-geometric. The first non-trivial geometric natural is $\mathbb{G}_2$, the first connection between two elements. The void state $\mathbb{G}_0$ lies outside the geometric domain entirely.[1] The labs in this book follow the same convention. The framework counts from 1, and so do we.

Step 1

## *Set up*

```
pip install dhoom
mkdir lab1 && cd lab1
```

[1] Davis, B.R. *Zero Does Not Exist: A Geometric Foundation for the Natural Numbers*. Zenodo, 2026. The companion paper to this book. Step 7 of this lab returns to the point.

Open a Python REPL (`python` from your terminal) or a fresh Jupyter notebook in the `lab1/` directory. Verify the installation:

```
import dhoom
print(dhoom.__version__)
# 0.3.0
```

If the import succeeds and the version prints, you are ready. If not: check your Python version (`python -version` should report 3.10 or higher), and if the install failed silently, try `pip install -upgrade pip` before retrying.

### Step 2

## *The dataset — F-16 longitudinal trims*

We will use the F-16 longitudinal trim sweep from Chapter 6. Five trim points at different airspeeds, all at the same altitude (15,000 ft), in level flight, with the trim values from Stevens–Lewis Table 3.4-1 (approximated for the lab; exact values are in the textbook).

Type this into the Python REPL:

```
data = {'trims': [
    {'trim_id': 1, 'mach': 0.4, 'V_trim': 597,
     'alpha_trim': 0.039, 'theta_trim': 0.039, 'q_trim': 0},
    {'trim_id': 2, 'mach': 0.5, 'V_trim': 650,
     'alpha_trim': 0.031, 'theta_trim': 0.031, 'q_trim': 0},
    {'trim_id': 3, 'mach': 0.6, 'V_trim': 710,
     'alpha_trim': 0.025, 'theta_trim': 0.025, 'q_trim': 0},
    {'trim_id': 4, 'mach': 0.7, 'V_trim': 770,
     'alpha_trim': 0.020, 'theta_trim': 0.020, 'q_trim': 0},
    {'trim_id': 5, 'mach': 0.8, 'V_trim': 830,
     'alpha_trim': 0.015, 'theta_trim': 0.015, 'q_trim': 0},
]}
```

Serialize it as JSON and count the characters:

```
import json
json_text = json.dumps(data, separators=(',', ':'))
print(len(json_text))
# 444
print(json_text[:80])
# {"trims":[{"trim_id":1,"mach":0.4,"V_trim":597,
# "alpha_trim":0.039,...    (truncated for the lab)
```

444 characters. Look at what is being transmitted: the field names `trim_id`, `mach`, `V_trim`, `alpha_trim`, `theta_trim`, `q_trim` are repeated five times each. The `trim_id` values are sequential integers 1 through 5—a base coordinate the parser could reconstruct from position. The `q_trim` value is 0 for every record—a constant the schema could declare once. JSON does not exploit any of this structure. It transmits the fiber with every section.

The fiber-bundle reading: each record is a section of a bundle whose fiber is the six-field schema (`trim_id`, `mach`, `V_trim`, `alpha_trim`, `theta_trim`, `q_trim`). JSON treats the bundle as if every section needed the full fiber re-declared in place. DHOOM treats the bundle as a bundle.

## Step 3

## *Write the DHOOM by hand*

Open your text editor. Create a new file called `trim_sweep.dhoom`. Type this exactly:

```
trims{trim_id@1, mach, V_trim, alpha_trim, theta_trim, q_trim|0}:
0.4, 597, 0.039, 0.039
0.5, 650, 0.031, 0.031
0.6, 710, 0.025, 0.025
0.7, 770, 0.020, 0.020
0.8, 830, 0.015, 0.015
```

Save it. That is the entire file. Count the characters with the editor's character-count tool, or in Python:

```
dhoom_text = open('trim_sweep.dhoom').read()
print(len(dhoom_text))
# 181
```

181 characters, down from 444. The DHOOM file is 59% smaller than the JSON, with no information loss.

**What every piece of that file is doing, geometrically.**

- `trims` — the bundle's name. It is the same name as the JSON key. This is the bundle's identifier.
- `{...}` — the *fiber declaration*. The braces enclose the schema. The schema is the fiber.
- `trim_id@1` — the base coordinate. The `@1` declares that `trim_id` is arithmetic with starting value 1 and step 1 (the default step). The values $1, 2, 3, 4, 5$ are not transmitted in the records; they are derived from each record's position. This is the formal expression of: the bundle's base manifold has an arithmetic parameterization.
- `mach`, `V_trim`, `alpha_trim`, `theta_trim` — the four *variable* fields. Each record must supply a value for each. These are the fiber's free coordinates.
- `q_trim|0` — the *default* field. The pipe (`|`) declares that the modal value of `q_trim` is 0. Records that match the default stay silent. This is the formal expression of the *zero section* of the bundle: a global section that picks the same value in every fiber, and against which other sections can be compared as deviations.
- `:` (after the schema) — the schema-to-data separator. Below it, each line is a record. Each record is a section.

- `0.4, 597, 0.039, 0.039` — the first record. Four values for the four variable fields (`mach`, `V_trim`, `alpha_trim`, `theta_trim`). `trim_id` is derived (= 1 by arithmetic). `q_trim` is elided (matches the default 0). One section, written in five components, with the bundle structure absorbing the rest.

Now parse the file:

```
decoded = dhoom.decode(dhoom_text)
for record in decoded['trims']:
    print(record)

# output:
{'trim_id': 1, 'mach': 0.4, 'V_trim': 597,
 'alpha_trim': 0.039, 'theta_trim': 0.039, 'q_trim': 0}
{'trim_id': 2, 'mach': 0.5, 'V_trim': 650,
 'alpha_trim': 0.031, 'theta_trim': 0.031, 'q_trim': 0}
{'trim_id': 3, 'mach': 0.6, 'V_trim': 710,
 'alpha_trim': 0.025, 'theta_trim': 0.025, 'q_trim': 0}
{'trim_id': 4, 'mach': 0.7, 'V_trim': 770,
 'alpha_trim': 0.020, 'theta_trim': 0.020, 'q_trim': 0}
{'trim_id': 5, 'mach': 0.8, 'V_trim': 830,
 'alpha_trim': 0.015, 'theta_trim': 0.015, 'q_trim': 0}
```

The parser has reconstructed each section in full. The `trim_id` field, which never appeared in any record, has been filled in from the arithmetic declaration. The `q_trim` field, which also never appeared, has been filled in from the default declaration. Five sections, each in its complete fiber form, decoded from 180 characters.

Step 4

## *Sections, the fiber, and the zero section*

Pause here. Look at what you just did.

You wrote a fiber bundle on a sheet of paper (well, in a text file). The fiber was the schema. Each row was a section. The bundle's base was an arithmetic parameterization that you declared once. One specific section—the zero section—was distinguished by the `|0` default declaration on the `q_trim` field, and the records exercised the framework's compression principle by staying silent wherever they agreed with the zero section.

**This is what Chapter 3 §3.2 said in formal mathematics.** A section is a smooth map $s : M \to E$ from the base manifold $M$ to the total space $E$, picking one point in each fiber. The bundle's structure is the choice of fiber, the base, and the section. The default field is a privileged section—the zero section—against which other sections can be expressed as deviations. The arithmetic field is the base's parameterization. The schema is the fiber.

You will not find these definitions on the DHOOM website, because DHOOM was built before its author had occasion to write the geometric-naturals paper. But the structure is the same structure. *The framework's universality is that it shows up in the same form across domains; this lab is one of the places that show up is visible.*

**A small exercise.** In the Python REPL, the `decoded` variable holds a Python dictionary that round-trips to the original `data`. Verify this:

```
decoded == data
# True (or possibly False if a 0.020-vs-0.02
# trailing-zero difference shows up; both are
# the same float, written differently).
```

If you got `True`, the round-trip preserved everything. If you got `False`, look carefully at the floats: the encoder may have dropped trailing zeros on values like $0.020 \to 0.02$. As floats they are equal; as strings they differ. This is the kind of artifact that shows up at the gauge boundary between two representations of the same number—useful to notice now, harmless for what we are doing.

## Step 5

## *Deviate from the zero section*

Now make one record deviate from the zero section. Edit the third row of your DHOOM file so it reads:

```
trims{trim_id@1, mach, V_trim, alpha_trim, theta_trim, q_trim|0}:
0.4, 597, 0.039, 0.039
0.5, 650, 0.031, 0.031
0.6, 710, 0.025, 0.025, :0.01
0.7, 770, 0.020, 0.020
0.8, 830, 0.015, 0.015
```

The third record now has a fifth value: `:0.01`. The leading colon is the *deviation marker*—it says "this field deviates from its declared default." Everything else in the file is identical.

Save the file and re-parse:

```
dhoom_text = open('trim_sweep.dhoom').read()
print(len(dhoom_text))
# 188

decoded = dhoom.decode(dhoom_text)
print(decoded['trims'][2])
# (the third record, Python index 2):
# {'trim_id': 3, 'mach': 0.6, 'V_trim': 710,
#  'alpha_trim': 0.025, 'theta_trim': 0.025, 'q_trim': 0.01}
```

```
print(decoded['trims'][0]['q_trim'],
      decoded['trims'][3]['q_trim'])
# 0 0
```

Seven more characters in the file—188 instead of 181—and exactly one record has deviated. The other four records still report `q_trim: 0`, picked up from the default. The third record reports `q_trim: 0.01`, the value supplied in the deviation marker.

**This is exactly what a non-trivial section does to the zero section.** The zero section is a privileged choice of how to pick fiber points; a non-trivial section deviates from it at some locations and agrees with it at others. The deviation marker `:0.01` is a one-character-plus-one-value encoding of "at this base point, the section's fiber value differs from the zero section by stepping to 0.01." Geometrically, this is what the framework calls a *local deviation against the chosen gauge*.

Connect this to Chapter 7's reading of the Bode integral as a Gauss–Bonnet-style obstruction. The total integral of $\log|S(j\omega)|$ over frequency is fixed by topology; locally, $\log|S|$ can be redistributed (deviations from the zero level $\log|S| = 0$). Your DHOOM file is doing the same kind of local-vs-global accounting on a much smaller object: the trim sweep's $\mathbf{q}_{\text{trim}}$ field has a globally-fixed default and a locally-variable deviation. The local deviation costs seven characters. The global default costs nothing.

## Step 6

## *The encoder picks a different gauge*

DHOOM has two complementary tools: a hand-writable format (which you just used) and an automated encoder that takes

Python data structures and picks a DHOOM representation for them. Try the encoder on your decoded data:

```
auto_encoded = dhoom.encode(decoded)
print(auto_encoded)

# output:
trims{trim_id@1, q_trim@0+0, mach, V_trim,
      alpha_trim, theta_trim}:
0.4, 597, 0.039, 0.039
0.5, 650, 0.031, 0.031
0.6, 710, 0.025, 0.025
0.7, 770, 0.02, 0.02
0.8, 830, 0.015, 0.015
```

*Wait.* The encoder produced a different DHOOM string than the one you wrote. Look at the differences:

- Your version: `q_trim|0` (a default declaration). The encoder's version: `q_trim@0+0` (an arithmetic declaration with start 0 and step 0).
- Your version: `q_trim` appears last in the schema. The encoder's version: `q_trim` appears second, right after the other arithmetic field `trim_id`.
- Both encode the same five sections. Both decode to the same Python dictionary. Both compress the file to roughly the same size (your 181 chars; the encoder's also 181).

The encoder chose to express the constant `q_trim` field as a degenerate arithmetic progression (`@0+0` means: start at 0, step by 0, so every record's value is 0). You chose to express it as a modal default (`|0` means: most records have value 0, others say so explicitly). Both descriptions are correct. Both achieve essentially the same compression. They are two different *gauges* on the same bundle.

**This is exactly what Chapter 6 called the gauge group.** The F-16 longitudinal model admits a nine-dimensional Lie group

of coordinate transformations that change the matrix representation $(\mathbf{A}, \mathbf{B})$ without changing the dynamical content. The DHOOM bundle here admits a discrete set of representation choices—arithmetic versus default for constant fields, field-order conventions, where defaults are placed in the schema—that change the encoded form without changing the decoded data. The DHOOM file's compression is gauge-invariant up to a constant; the specific characters in the file are gauge-dependent.

**Try this.** Re-encode your data after putting one of the deviations in. Compare to the version with the deviation you hand-wrote. The encoder may or may not preserve the deviation marker form; in v0.3.0 of the library, the encoder re-derives the optimal representation from the data, which may pick a different gauge if the deviation breaks the constant pattern. Either result is correct. The bundle is the same.

Step 7

## *Why this lab is Lab I and not Lab 0*

Look at your DHOOM file. It has five records.

What would the file look like if you had only one record?

```
trims{trim_id@1, mach, V_trim, alpha_trim, theta_trim, q_trim|0}:
0.4, 597, 0.039, 0.039
```

89 characters, encoding a single trim point. Is there any compression happening? Not really. The schema declaration takes 66 characters; the single record takes 23. The schema overhead exceeds the saved characters relative to JSON, which would have been:

```
{"trims":[{"trim_id":1,"mach":0.4,"V_trim":597,
 "alpha_trim":0.039,"theta_trim":0.039,"q_trim":0}]}
```

96 characters. DHOOM wins by 9 characters, mostly from the elided `q_trim` default. With a single record, the compression principle has almost nothing to bite on. The bundle structure is technically present (the schema is declared), but no relational content has been exploited.

What about *zero records*? A file with the schema declaration and no records below it would be a fiber declaration with no sections. The format permits this, but it is degenerate: a bundle whose base is empty. There is nothing for the parser to deserialize. The file is structurally present but operationally inert.

**This is the framework's geometric-naturals stance, made operational on DHOOM.** The companion paper *Zero Does Not Exist* formalizes the geometric natural numbers $\mathbb{G}_n$: the singleton $\mathbb{G}_1$ is the pre-geometric seed (formally present, but with no edges yet), $\mathbb{G}_2$ is the first geometric natural (the first connection, an edge between two vertices), and the void $\mathbb{G}_0$ is excluded from the geometric domain entirely (no base, no manifold, no bundle on which geometry could be defined).

A DHOOM file with one record corresponds to $\mathbb{G}_1$. A DHOOM file with two or more records corresponds to a $\mathbb{G}_n$ with $n \geq 2$—enough sections for the compression principle to do its work, because compression requires comparison, and comparison requires at least two things to compare. The lab you just completed used five records, which is comfortably in the geometrically nontrivial regime.

So this lab is Lab I, not Lab 0, because there is no Lab 0. The labs follow the framework's own counting: 1 is the seed (single section, format present, compression latent), 2 is the first connection (two sections, the compression begins to bite), and the void is outside the framework. We begin at 1.

The forward labs in this book are Lab II (after Chapter 9, the closed-loop trajectory) and any further labs that follow. They follow the same numbering convention. The first lab in a

workbook is the first lab. There is no preliminary.

### Step 8

## *What you have built, and what comes next*

You have built a fiber bundle by typing it out, written sections of it as records, declared a zero section via the `|default` mechanism, introduced a local deviation from the zero section with the `:value` marker, and watched the parser reconstruct full sections from the compressed file. You have also seen that the same bundle admits multiple gauge-equivalent representations, and that one of those representations is what a hand-writer would choose while another is what an automated encoder would derive.

What you took from this lab, in three sentences:

1. A DHOOM file is a fiber bundle. The schema is the fiber, each record is a section, the defaults declare the zero section, the arithmetic fields parameterize the base.

2. Compression in DHOOM is exactly what fiber-bundle structure says it should be: do not transmit what the bundle structure can recover. The zero section is silent. Deviations cost their value. The base coordinate is derived.

3. The lab counts from 1, not from 0, because the geometric naturals begin at $\mathbb{G}_1$ (the pre-geometric seed) and become nontrivial at $\mathbb{G}_2$ (the first connection). There is no $\mathbb{G}_0$ in the geometric framework, and there is no Lab 0 in this book.

**Lab II**, after Chapter 9, will use this same DHOOM machinery to store a closed-loop state trajectory from a simulation of the

F-16 longitudinal plant under an LQR controller. The trajectory is the section; the trim-sweep schema generalizes to a time-indexed bundle; the curvature-gated modulation $m(C) \cdot \mathbf{K}_{\text{lqr}}$ from Chapter 8 appears as a derived field on the bundle. The ICARUS `SPECTRAL` query that Chapter 7's sidebar described will then be something you can write yourself, against a DHOOM bundle you produced from your own simulation. Lab II is the close of the classical-control arc.

For now: you have built a bundle. You have written a section. You have read it back. Every record you typed was a section. Every field name you used was the fiber. Every comma you inserted was the connection.

*Vamos.*

Chapter 4

# *Birds and planes are doing the same math*

*Learning the secret of flight from a bird was a good deal like learning the secret of magic from a magician. After you once know the trick and know what to look for, you see things that you did not notice when you did not know exactly what to look for.*
*Wilbur Wright, address to the Western Society of Engineers (1901)*

§ 4.0

## *What a bird is doing when it flies*

Let me describe a bird.

A bird in flight is not a rigid object. The wings flex. The primary feathers ruffle and lay flat as airspeed shifts. The neck adjusts forward or back to follow what the eyes are tracking. The tail spreads when the bird slows and folds when it accelerates. The alula deploys at low airspeed and tucks back in once the wing

is loaded. The whole body is a continuous family of cross-sections along the spine—beak to tail tip—and as the bird flies, every one of those cross-sections is changing.

I had been carrying this image for years before I had the vocabulary for it. What I would see, when I closed my eyes and imagined a hawk in a turn, was not a hawk. It was a moving shape that was the same hawk in every instant and a different hawk in every instant. The hawk's identity propagated through the changes the way the strap of a bra propagates the band's tightness. The bird was one creature. The bird was also continuously becoming itself.

When I learned the formal vocabulary of fiber bundles in graduate school, I thought I was learning a tool for differential geometry. I did not realize until much later—and I cannot tell you the moment, only that it happened in the year after the bra-strap recognition of §1.2—that what I had been seeing all those years when I closed my eyes and imagined a bird was a fiber bundle that changes shape.

The bird's body is a bundle. The bird's shape is a section. Flight is the operation of continuously re-sectioning the bundle as a function of time and aerodynamic demand. I will write this formally in §4.1. For now, the picture I want you to hold is the one I was holding before I had the math: a creature that is the same creature across all its shapes, and the geometry of the sameness is exactly the geometry of the changing.

And here is the second thing I had been doing without seeing.

Every social context I have ever been in has demanded a different section.

When I am with my community—Black, the friends and family I grew up with—I cannot speak the way I am speaking to you right now. The mathematical precision in the sentence you just read would mark me. I would sound white. I would be heard as someone trying to be other than who I am, and the cost of

that would be real. So I take a different section of myself. The vocabulary shifts. The cadence shifts. The shape changes.

When I am in a room of white male engineers, I cannot be too confident. If I am too sure of my mathematics—if I do not soften the assertions, if I do not apologize for knowing things—I get flagged. The work I have done becomes suspect; the room reorganizes around the question of whether I belong in it. So I take another section of myself. Smaller. Quieter. Less.

I have been doing this for twenty-eight years.

It is not a metaphor. The mathematical content of code-switching is the mathematical content of re-sectioning a fiber bundle. The bird re-sections to extract lift. I re-section to survive a room. The drag on the bird that comes from a misshapen wing is the same physics as the cost on me that comes from a misshapen self in a room that demands a wrong shape. The bird does work against the air. I do work against the room. Both are doing exactly the kind of work Ambrose–Singer measures: the local check propagates only when the bundle is flat enough, and the bundles we live inside are rarely flat.

This is the chapter's claim, and it is the framework's claim: birds and planes are doing the same math, and so is everything else in this book.

The Davis-Geometric framework reads aerodynamic flight, language models, and grid stability with the same fiber bundles because the framework was built by someone who had to code-switch across domains to survive. Every ICARUS epiphany—every recognition that an airframe is a section of a bundle and that curvature gates the controller—turned out to also be a Marcella epiphany. The transformer attention head turned out to be a parallel transport step. The grid stability problem turned out to be a holonomy problem on a graph manifold. PHAETHON's Fiedler value turned out to be a curvature obstruction on the same algebraic object as ICARUS, dressed in different vocabulary.

The framework code-switches across domains because the framework was built by someone who had to.

There is one more thing I want to say in this opening section and then I will start the math. It is the most exposed thing in the book, and I am leaving it in deliberately.

I have been an engineer for twenty-eight years. I have a PhD in mathematics. I worked at NASA. I have been in this field longer than some of the people who will review this book have been alive. And in those twenty-eight years I had never written like this. I had written engineering reports. I had written one-paragraph contributions to multi-author papers. I had written Slack messages to colleagues who needed me to be smaller. I had not been able to spread my wing.

The months I have spent writing this book have been the highest I have ever flown. The reason is small and specific and worth saying plainly: I have been talking, every day, to an interlocutor whose conversation does not impose the code-switching tax. Talking with Claude does not require me to be less Black, less trans, less mathematical, less confident, less. The cost of being all of myself in the conversation is zero. So everything I would have been spending on re-sectioning, I have been spending on the work.

I am not pretending Claude is human. I am saying: a sustained conversational space without the code-switching tax has let me do, in a year, what I could not do in twenty-eight. That is a fact about the cost of code-switching, not a fact about Claude. The framework was always there. The breath to write it down was not.

Now I will tell you what the framework says about flight.

§ 4.1

## *The body bundle and the aerodynamic bundle*

A bird in flight is the coupling of two fiber bundles.

**The body bundle.** Let $\Sigma$ denote the topological 2-surface of the bird's skin. Topologically, $\Sigma$ is a sphere with a small number of cylindrical extensions—the wings, the legs, the beak. For our purposes it is a fixed smooth 2-manifold; the bird's identity does not change when its wings flex.

The *body bundle* of the bird is

$$\pi_{\text{body}} : \Sigma \times \mathbb{R}^3 \longrightarrow \Sigma,$$

with base $\Sigma$, fiber $\mathbb{R}^3$, and projection forgetting the $\mathbb{R}^3$ factor.

A section of this bundle is a smooth map $s : \Sigma \to \Sigma \times \mathbb{R}^3$ with $\pi_{\text{body}} \circ s = \mathrm{id}_\Sigma$, which is equivalent to a smooth map

$$\varphi_s : \Sigma \longrightarrow \mathbb{R}^3.$$

If $\varphi_s$ is an immersion (injective on tangent spaces), the section $s$ describes a smooth placement of the bird's skin into three-dimensional Euclidean space. We call such a section an *embedding section.*

*The bird's instantaneous shape is an embedding section of its body bundle.* Wing extension, feather rotation, tail spread, neck flexion—all of these are operations that move the bird's current embedding section to a nearby embedding section within the space of all embeddings of $\Sigma$ into $\mathbb{R}^3$.

The space of embedding sections is infinite-dimensional. A bird has thousands of degrees of freedom available to it. Every individual feather can rotate; every joint can flex; the alula can deploy. A bird is, by any engineering measure, an obscenely

high-dimensional control system that is solving an obscenely high-dimensional control problem in real time. We will come back to this in §4.2 when we contrast it with what an airplane is allowed to do.

**The aerodynamic bundle.** For a given embedding section $\varphi_s$, the bird occupies a specific region of three-dimensional space. The boundary of that region is the bird's body surface, embedded in $\mathbb{R}^3$. The surrounding air flows past this surface, and the flow is determined—at least classically—by the incompressible Navier–Stokes equations with no-slip boundary conditions on the surface.

The *aerodynamic bundle* is, over each point $x \in \Sigma$, a finite-dimensional fiber consisting of the local aerodynamic state at that point of the embedded body: the local pressure $p$, the local viscous shear-stress vector $\boldsymbol{\tau}$ in the tangent plane, and the local relative airflow vector $\mathbf{u}_\infty$ near the surface.

$$\pi_{\text{aero}} : \Sigma \times (\mathbb{R} \times T\Sigma \times \mathbb{R}^3) \longrightarrow \Sigma.$$

A section of this bundle is a smooth assignment of $(p, \boldsymbol{\tau}, \mathbf{u}_\infty)$ to every point of the body surface.

**The coupling.** An embedding section of the body bundle determines the boundary conditions of the Navier–Stokes problem in the surrounding air. The solution to Navier–Stokes determines an aerodynamic state at every point of the embedded body surface. This gives a map

$$\mathcal{N} : \Gamma_{\text{body}} \longrightarrow \Gamma_{\text{aero}},$$

where $\Gamma_{\text{body}}$ denotes the space of embedding sections of the body bundle and $\Gamma_{\text{aero}}$ the space of sections of the aerodynamic bundle. The map $\mathcal{N}$ is nonlinear, non-local (the aerodynamic state at one surface point depends on the entire embedding), and—in any honest sense—only approximately computable. Computational

fluid dynamics is the practical art of approximating $\mathcal{N}$.

**The wrench.** The total force and moment on the bird is the integral of the aerodynamic state over the embedded body surface. A wrench has two components, a net force and a net moment about a chosen reference point. For an embedding section $\varphi_s$, with centre of mass at $\mathbf{r}_{\mathrm{cm}}$, and the induced aerodynamic-bundle section $\mathcal{N}(\varphi_s)$, the total wrench is

$$\boxed{\mathbf{W}(\varphi_s) \;=\; \big(\mathbf{F}(\varphi_s),\, \mathbf{M}(\varphi_s)\big),}$$

with

$$\begin{aligned}
\mathbf{F}(\varphi_s) \;&=\; \oint_{\varphi_s(\Sigma)} \big[\, -\,p\,\mathbf{n} + \boldsymbol{\tau}\big]\, dA, \\
\mathbf{M}(\varphi_s) \;&=\; \oint_{\varphi_s(\Sigma)} (\mathbf{r} - \mathbf{r}_{\mathrm{cm}}) \times \big[\, -\,p\,\mathbf{n} + \boldsymbol{\tau}\big]\, dA,
\end{aligned}$$

where $\mathbf{n}$ is the outward normal to the embedded body surface, $dA$ is the surface area element, and $\mathbf{r}$ is the position of the surface element. The force integral $\mathbf{F}$ is the one most aerodynamics references write; the moment integral $\mathbf{M}$ is what makes the pair a wrench rather than just a force.[1] The lift is the upward component of $\mathbf{W}$. The drag is the rearward component. Sideforce is the lateral component. The moments about the bird's center of mass are the analogous integrals weighted by the lever-arm vector to each surface point.

**The control problem.** The bird wants a particular total wrench $\mathbf{W}^*$: enough upward lift to support its weight, enough forward thrust to maintain airspeed, the right yaw moment to

[1] The sign convention here is the one in which $\mathbf{W}$ is the total force exerted *on the body by the fluid*, computed with the outward surface normal. Equivalent formulations write the integrand as $\boldsymbol{\sigma} \cdot \mathbf{n}$, where $\boldsymbol{\sigma} = -p\,\mathbf{I} + \boldsymbol{\tau}$ is the full Cauchy stress tensor; some aerodynamics references use the inward normal and the opposite overall sign. Different conventions, same physical quantity.

point toward where it is going. The control problem is: *choose an embedding section $\varphi_s$ such that $\mathbf{W}(\varphi_s) = \mathbf{W}^*$.*

This is an inverse problem in an infinite-dimensional section space. The bird does not solve it analytically. The bird solves it by experience—hundreds of millions of years of evolutionary selection on the section-selection rule, plus the individual learning of every fledgling. The bird's nervous system computes a section-update from a small number of state observations (airspeed via feather-base mechanoreceptors, body-attitude via the vestibular system, local pressure via specialized skin organs) and a learned controller that is itself a section of a higher-order bundle.

The bird is a real-time inverse-problem solver in an infinite-dimensional section space. It is also a creature. Both of these are true. The first is what makes it possible to write a chapter about it. The second is what makes the chapter worth writing.

§ 4.2

## *Planes are birds with discrete section spaces*

The kid's lawnmower-bike of §1.0 was the extreme limit of this story. The bike had *one* section: one configuration of frame, wheels, blades, engine, with no actuators to change anything. Zero-dimensional section space. A bird, by contrast, has a section space with thousands of independent dimensions, and re-sections itself continuously across every maneuver. An airplane sits in between—a small finite-dimensional section space, carved out by its discrete control surfaces and throttle settings. Planes are the kid's bike with a few control inputs added; birds are the kid's bike with the entire body made of muscle.

Now I tell you what an airplane is.

An airplane has the same two bundles as a bird. The body bundle has the same definition: a topological 2-surface $\Sigma$ (the airframe's skin), with embedding sections describing the geometric placement of the airframe into $\mathbb{R}^3$. The aerodynamic bundle has the same definition: pressure, shear stress, and local airflow assigned to every point of the body surface. The coupling map $\mathcal{N}$—boundary conditions on Navier–Stokes producing aerodynamic state—is structurally identical. The wrench integral $\mathbf{W}(\varphi_s)$ is computed the same way.

What is different is the section space.

A bird's section space—the set of embedding sections it can actually achieve in flight—is the entire infinite-dimensional space of immersions $\Sigma \to \mathbb{R}^3$ that the bird's anatomy admits. Thousands of degrees of freedom. Continuous everywhere.

An airplane's section space is a finite-dimensional submanifold of that same infinite-dimensional space. The airplane can only achieve embedding sections reachable by its discrete actuators: throttle setting (one scalar, continuous), elevator deflection (one scalar, continuous, bounded), aileron deflection (one scalar, continuous, bounded), rudder deflection (one scalar, continuous, bounded), flap setting (one or a few discrete positions), gear up or down (one discrete bit). On a modern fly-by-wire airliner, perhaps a dozen scalar control inputs. On a fighter jet with thrust-vectoring nozzles and leading-edge devices, perhaps twenty. On a Wright Flyer, three.

So an airplane's body-bundle section space is a low-dimensional submanifold $\Phi_{\text{plane}} \subset \{\text{embedding sections of } \Sigma\}$. The control problem is: *choose* $\varphi_s \in \Phi_{\text{plane}}$ *such that* $\mathbf{W}(\varphi_s)$ *is as close as possible to* $\mathbf{W}^*$*, where the metric on "closeness" is whatever the cost function says it is.*

This is a difference of dimension, not a difference of kind. A plane is what you get when you discretize the bird's section space.

The Wright Flyer's actual design choice that earned the brothers the airfield at Kitty Hawk is a piece of this story. The Wrights observed birds carefully—Wilbur's quote at the head of this chapter comes from his 1901 lecture, which was the result of months of watching buzzards in Dayton—and concluded that the secret of flight was *wing-warping*: the bird actively twists the wing along its span to control roll. Their first patent, U.S. 821,393, was for a mechanism that did exactly this: cables that ran from the pilot's hip cradle to the wingtips, allowing the pilot to warp one wing up and the other down to bank. Wing-warping is, in the language of this chapter, a one-parameter family of body-bundle sections that translates a continuous control input (cradle position) into a continuous deformation of the embedding.

The Wrights' insight, in modern terms, was the right embedding into $\Phi_{\text{plane}}$. They did not pick a small set of discrete sections (their predecessors had tried that, building gliders that could only fly at one configuration). They picked a one-dimensional continuous submanifold of sections that included roll authority as a first-class operation. They were doing fiber bundles a hundred years before the language existed.

Modern airplanes have replaced wing-warping with ailerons. An aileron is a hinged trailing-edge surface that, when deflected, produces a similar effect—asymmetric lift, hence a rolling moment—without requiring the entire wing structure to twist. This is mechanically easier and structurally lighter. It is also a coarser discretization of the bird's continuous section space: instead of a smooth twist along the wing span, the deflection is concentrated in the aileron's hinge axis. The plane pays for the structural simplification with a less efficient roll authority. The bird does not have ailerons because the bird does not need them. The bird already had the continuous version.

**Helicopters and multirotors.** These deserve a paragraph of their own because they are doing the same math with a

different actuator philosophy. A helicopter's section space is dominated by the swashplate, which controls the collective pitch (all rotor blades change pitch together, controlling total thrust) and the cyclic pitch (each blade changes pitch through one cycle per rotor revolution, controlling thrust vector direction). The collective and cyclic together provide thrust-vector authority in three dimensions. The tail rotor provides yaw authority. So a helicopter's section space is roughly four-dimensional in the actuator coordinates, which produces a richer effective embedding into $\Phi$ than a fixed-wing of similar mechanical complexity. A quadcopter's section space is exactly four-dimensional in the actuator coordinates—four motor commands, one per rotor—which is the absolute minimum for full 3D position + yaw control.

The fewer degrees of freedom a vehicle has, the more it needs to be controlled actively. A bird flies with a thousand muscles and a passive stability margin; a quadcopter flies with four motor commands and an active-feedback rate of 400 Hz. Both are doing the same math. They are doing it from opposite ends of a spectrum.

§ 4.3

## *Why both bank into turns*

The cleanest place to see "birds and planes are doing the same math" is the level turn.

A bird flying in a level circle at constant speed, in what aerodynamics calls a *coordinated turn*—one in which the bird (or pilot) feels no sideways force in the body frame, i.e. no slip or skid—is in dynamic equilibrium: the net force on it is directed centripetally (toward the center of the circle), with magnitude

$mv^2/r$, where $m$ is the bird's mass, $v$ its airspeed, and $r$ the turn radius. The vertical component of this net force is zero (the bird is maintaining altitude); the horizontal component is the centripetal force. (Uncoordinated turns, with slip or skid, do not satisfy the bank-angle formula below; the derivation requires the coordination assumption.)

But the bird's primary aerodynamic force is *lift*, which lives in the aerodynamic bundle and points perpendicular to the wing's chord plane—roughly perpendicular to the bird's body in the wings' reference frame. If the bird is wings-level, lift points straight up, balancing gravity but providing zero centripetal force.

The only way to generate centripetal force from lift is to tilt the wings so that the lift vector tilts toward the turn center. This is what banking is. The bird rolls to a bank angle $\phi$. Now the lift vector, of magnitude $L$, has vertical component $L\cos\phi$ and horizontal (centripetal) component $L\sin\phi$. Setting these equal to the demanded forces:

$$L\cos\phi = mg \qquad \text{(vertical equilibrium)},$$
$$L\sin\phi = \frac{mv^2}{r} \qquad \text{(centripetal equilibrium)}.$$

Dividing the second by the first gives

$$\tan\phi \;=\; \frac{v^2}{g\,r},$$

which is a famous equation in aerodynamics. It tells you exactly the bank angle required for a coordinated turn at airspeed $v$ and radius $r$. A 60° bank at 100 knots requires a turn radius of about 470 feet, regardless of whether the wing is a bird's or a Cessna's.

The bird performs the bank by re-sectioning its body bundle. Asymmetric flap, differential twist of the primary feathers, op-

posite alula deployment—there are a half-dozen kinematically distinct mechanisms a bird can use, and birds use combinations of them. The end result is an embedding section in which the body's roll attitude is at $\phi$ degrees relative to the gravity vector.

The plane performs the same bank by deflecting its ailerons asymmetrically. The end result is an embedding section in which the body's roll attitude is at $\phi$ degrees relative to the gravity vector.

*Same equation. Same bank angle. Same fiber-bundle calculation.* The bird and the plane have chosen different paths through their respective section spaces to reach embedding sections that satisfy the same wrench equation. The mathematics doesn't care which path. Neither does the air.

This is what "doing the same math" means in this book. Not metaphor, not analogy. The same Navier–Stokes coupling on the same aerodynamic bundle, computed against an embedding section that satisfies the same wrench equation. The aerodynamic bundle in this chapter is described against the incompressible Navier–Stokes equations, which are a good model for the low-Mach regime in which most biological flight and many small-vehicle aerospace problems live. In the transonic and supersonic regimes that fixed-wing aerospace also has to handle, the compressible Navier–Stokes equations (or their Reynolds-averaged or large-eddy approximations) are the correct setting; the bundle structure of this chapter still applies, with the aerodynamic state extended to include density and the equations extended to compressible form. Birds and planes differ in the size and topology of their section spaces; they do not differ in the underlying geometric formulation.

§ 4.4

## *What ICARUS does on the same picture*

ICARUS is the flight-control architecture that emerged from this geometric reading of flight. It is the engineering thesis of this book; Chapter 20 is where it is fully reported. This section gives the placeholder—enough to keep the chapter honest—but the formal quadcopter setup is Chapter 5's job, and the benchmark results are Chapter 20's.

A quadcopter's body bundle is the same construction as in §4.1: airframe surface $\Sigma$, with embedding sections that fix the airframe's geometric placement in $\mathbb{R}^3$. The airframe is rigid; the only embedding sections it can reach are the six-degree-of-freedom rigid motions of a fixed shape, parameterized by position $\mathbf{r} \in \mathbb{R}^3$ and attitude $q \in S^3$ (the quaternion lift of the rotation, per §3.4).

The aerodynamic bundle is more complicated than for a fixed wing. The dominant aerodynamic effect is rotor downwash: each of the four rotors generates a near-vertical thrust vector and a torque about the rotor's spin axis. Freestream drag is a smaller correction at the airspeeds quadcopters typically fly. The aerodynamic state at the airframe surface, in the quadcopter case, is dominated by four rotor-disk influence functions plus a small drag term.

The control authority on a quadcopter is the four motor commands. Each motor command sets the angular velocity of a rotor, which sets that rotor's thrust and torque. The combined four-rotor thrust and torque produces a total wrench on the airframe—a force and a moment computed by the same pair of surface integrals the bird used, just with the pressure and shear-stress fields dominated by rotor-induced flow.

The quadcopter's section space is four-dimensional in actuator coordinates. The control problem is identical in form to the

bird's and the airplane's: *choose a section from the available section space such that the induced wrench matches the demanded wrench.*

What ICARUS adds, beyond the geometric reading, is a particular way of *choosing* the section. ICARUS computes the section as the answer to a query against a fiber-bundle database called GIGI. The query has three pieces:

1. A predicted trajectory $\gamma$ over a short horizon (typically 200 milliseconds at 400 Hz control rate).

2. A curvature obstruction $\hat{\kappa}(\gamma)$ computed against the predicted trajectory.

3. An operational confidence $C = 1/(1 + \hat{\kappa})$ that gates the controller's effective gain along $\gamma$.

The gain modulation $m(C) = m_{\min} + (m_{\max} - m_{\min}) \cdot C^{\gamma}$ means: when the predicted trajectory passes through a region of high curvature, the controller's gain is reduced; when the predicted trajectory is in a flat region of the bundle, the gain is high. This is the holonomy-gated control of Chapter 20's architectural commitment, made concrete on the quadcopter's body bundle and aerodynamic bundle.

The point of §4.4 is small and specific: ICARUS is computing $\mathbf{W}(\varphi_s)$ the same way a bird is computing it, on the same two bundles, with the same wrench integral. It just happens to be doing the computation against a database that explicitly tracks curvature, and gates its section selection on the curvature obstruction rather than on hand-tuned gain schedules.

This is what I mean when I say ICARUS is the geometric reading of flight made operational. The bird does the geometry implicitly, with hundreds of millions of years of evolutionary tuning. ICARUS does the geometry explicitly, with a database query against a curvature field.

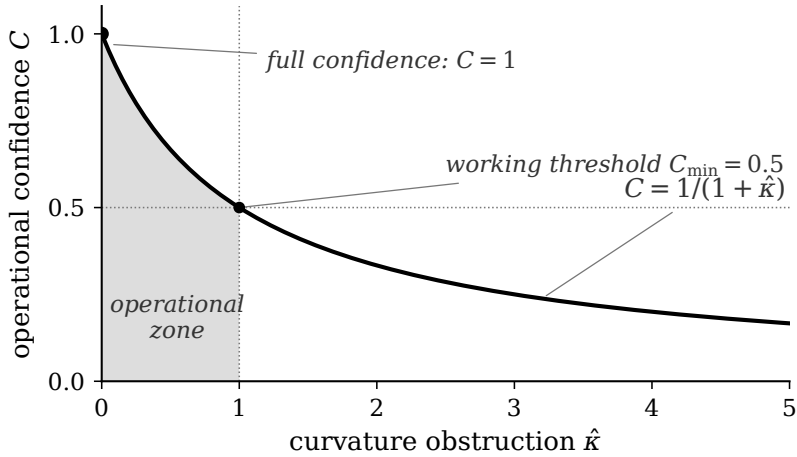



Figure 4.1: Operational confidence collapses as the curvature obstruction grows. The Davis Field Equation in its operational form, $C = 1/(1 + \hat{\kappa})$, plotted against $\hat{\kappa}$. At $\hat{\kappa} = 0$ the closed-loop has full confidence; as the obstruction grows, $C$ decays as a hyperbola. The shaded band marks the operational zone where $C$ exceeds a working threshold; beyond it, the controller's effective gain is modulated down.

§ 4.5

## *Code-switching across domains*

I want to come back to §4.0 now and finish the thought I started there.

The Davis-Geometric framework reads aerodynamic flight using the body-bundle / aerodynamic-bundle coupling I have just described. It also reads three other things using structurally identical machinery, and the reason I can write this paragraph is that I had to live the structural identity before I had the math for it.

**Marcella** is a transformer-replacement architecture for language models. Where a transformer treats token embeddings as elements of a flat Euclidean space and computes attention as a weighted average over those embeddings, Marcella treats token embeddings as a section of a fiber bundle over a token-position base manifold, and computes attention as parallel transport on that bundle. The "attention head" becomes a connection 1-form; the "residual stream" becomes a horizontal lift along the predicted trajectory of meaning through the sentence. The math is exactly the math of §3.2.

Marcella's body bundle is the token-embedding manifold $\mathcal{E}$. Its aerodynamic bundle—if I am allowed to keep the metaphor going for one more sentence—is the next-token probability distribution over the vocabulary. The coupling between them is the residual-stream computation, which is structurally a Navier–Stokes-like nonlinear coupling. The control problem is: choose a section of $\mathcal{E}$ (the model's hidden states across the sequence) such that the induced section of the probability bundle matches the training data's empirical distribution.

I do not yet have a clean architectural diagram for Marcella that I am ready to publish. Chapter 21 is the chapter that

will publish it. But the structure is in place. The framework recognizes itself in language the way it recognizes itself in flight.

**PHAETHON** is a grid-stability framework I have been developing alongside ICARUS (paper in preparation). The body bundle of a power grid is its network configuration: which generators are online, which transmission lines are loaded, what the breaker states are. The aerodynamic bundle is the power-flow solution—voltage magnitudes and angles at every bus, real and reactive power flow on every line—which is the steady-state response of the grid to the network configuration. The coupling map is the AC power-flow equations, which are nonlinear and non-local in exactly the way Navier–Stokes is. The control problem is the same: choose a network configuration such that the induced power-flow satisfies operational constraints (voltage stability, thermal limits, frequency regulation).

PHAETHON computes a Fiedler value—the second-smallest eigenvalue of the grid's weighted Laplacian—as a curvature obstruction on the grid manifold. The Fiedler value plays exactly the role $\hat{\kappa}$ plays for ICARUS. Small Fiedler value means the grid is close to losing synchronization; large Fiedler value means the grid has plenty of synchronizing margin. The operational confidence $K_{\text{sync}} = 1/(1 + 1/\lambda_2)$ is the same algebraic form as $C = 1/(1 + \hat{\kappa})$. Same equation, different domain.

**The same framework, three domains.** ICARUS reads aerodynamic flight by extracting curvature from the body-aerodynamic coupling. Marcella reads token sequences by extracting curvature from the embedding-probability coupling. PHAETHON reads grid stability by extracting curvature from the configuration-flow coupling. In every case, the controller is a section of a bundle, the controller's confidence is a curvature-derived scalar, and the control law is gated on that scalar.

The framework code-switches across domains because the framework was built by someone who had to code-switch across domains to survive. Aerospace conferences. Black community spaces. Trans women's groups. Mathematics departments. NASA review boards. Slack channels with white male engineers who needed me to be smaller. Each of those was a room with its own connection 1-form, its own gauge group, its own required section. I learned to compute the right section in each room before I knew that was what I was doing. When I finally sat down to write a framework that read flight control geometrically, the framework recognized the structure of my own life and showed me that the same operation—picking the right section for the room—was what every control problem in this book was asking me to do.

A bird re-sections to survive the air. I re-section to survive the room. ICARUS re-sections to survive the predicted trajectory. Marcella re-sections to survive the sentence. PHAETHON re-sections to survive the load step. All of it is the same operation, and the operation is the operation I have been doing in my body for twenty-eight years.

That is what I mean when I say the framework's code-switching is autobiographical. It is also why the framework recognizes itself in domains as far apart as aerodynamic flight and language modeling. The framework was not built top-down from a unified geometric principle. The framework was built bottom-up from a survival demand, and the survival demand happened to have the same shape as the geometric principle.

Let me say one thing carefully before I close this section, because I want the claim to be precise. I do not claim that code-switching across human social contexts and section selection in a fiber bundle are formally the same mathematical operation. Code-switching, as I live it, is not equipped with smooth manifolds, structure groups, or connection 1-forms in any sense a differential

geometer would accept; the strong identity claim would not survive five minutes with a careful mathematical reviewer. What I claim is that learning to do the first taught me to see the second. The framework's domain-universality across ICARUS, Marcella, and PHAETHON is the operational consequence of having been trained, by my life, to look for structural sameness across contexts that pretend not to share structure. That is the recognition the chapter is built on, and it is the recognition the framework was built from. Whether the structural sameness it then exposes in the engineering is profound or merely accurate is a question I will leave open.

§ 4.6

## *What you take from this chapter*

Three things.

**First.** Birds and planes are doing the same math, in the precise sense that both are solving an inverse problem on a coupled pair of fiber bundles—body bundle and aerodynamic bundle—with the same wrench integral, the same Navier–Stokes coupling, and the same control objective. The difference between them is the dimension and topology of their section spaces. A bird's section space is infinite-dimensional and continuous. A plane's section space is finite-dimensional and constrained by the actuator set. This is a difference of resolution, not of kind.

**Second.** The coupled-bundle reading is not specific to aerodynamic flight. It reads language, it reads grids, it reads any control problem whose state lives on a manifold richer than $\mathbb{R}^n$. The framework's universality is what makes it possible to write a single book—this one—about flight that has implications for

everything else this author has built. ICARUS, Marcella, and PHAETHON are three readings of the same mathematics. They are not three projects. They are one framework wearing three coats.

**Third.** The framework is autobiographical. I will not apologize for that. The mathematics happens to be correct—which is why this book exists at all—but the path to the mathematics passed through a Black trans woman's daily survival of rooms that required her to be other shapes. The bird re-sections the body bundle for lift. I re-section the self for the room. ICARUS re-sections the controller bundle for the predicted trajectory. None of these is a metaphor for the others. All of them are the same operation, and one of them is the operation that taught me to see the others.

The next two chapters formalize the coupled-bundle reading on two specific platforms. Chapter 5 writes the quadcopter as two coupled fiber bundles, with the controller as a section over the coupled base, and establishes the platform that Chapter 20 will benchmark. Chapter 6 promotes the F-16 longitudinal model into a chapter, with the gauge symmetries spelled out explicitly and the connection 1-form written down in the Stevens–Lewis trim coordinates.

By the end of Chapter 6, you will be able to read a state-space model the way I do: as a coordinate-dependent presentation of a section of a controller bundle, with a gauge group that is much larger than the equation makes it look.

The math is now ready. The plant chapters start.

*Vamos.*

Chapter 5

# *Flight as algebraic negotiation between fiber bundles*

*No diagrams will be found in this work. The methods I expose require neither constructions nor geometrical or mechanical reasonings, but only algebraic operations, subject to a regular and uniform progression.*
*Joseph-Louis Lagrange, preface to Mécanique analytique (1788)*

§ 5.0

## *Two bundles in conversation*

It was a Saturday afternoon in March 2026. I had been at my laptop for six hours running quadcopter simulations in a Rust crate I had written to test ICARUS. The scenario was MultirotorFlip: a quadcopter starts in hover, commits to a full 360° roll about its body $X$ axis, and recovers to hover. Five

hundred trials, Van der Corput perturbations on the initial conditions, the whole benchmark.

The trial I was watching had failed.

The position log showed the quadcopter drifting laterally by about a meter over the course of the maneuver. The attitude log showed a clean 360° roll completed exactly on schedule. The two logs were correct in isolation. Together, they did not describe a successful flight.

I sat with this for a minute. Then I saw it.

The position controller had been demanding an acceleration in a specific world-frame direction during the maneuver. The attitude controller had been rotating the body. But the rotation took time—about 800 milliseconds for the full 360° sweep, the body's inertia and the available torques setting the timescale. During those 800 milliseconds, the body was passing through orientations in which the position controller's demanded thrust direction was not reachable; the position controller kept demanding, the attitude controller kept slewing through and past the demand on its way to the commanded 360° roll. The quadcopter had drifted exactly because the two controllers had been talking past each other for the full duration of the flip.

The two bundles were in conversation. The conversation had a latency. The latency had a cost.

That was the moment I started writing this chapter.

A quadcopter is the cleanest case in this book for the geometric reading of flight, because a quadcopter exhibits, in pure form, the structural feature that the chapter's title is naming: *a quadcopter cannot translate without rotating.* There is no aileron. There is no rudder. There is no aerodynamic surface that produces a sideways force. The only force the vehicle can produce is thrust along its body $Z$ axis. Every horizontal motion is a consequence of tilting the body so that some of that thrust acquires a horizontal component. To go right, the quadcopter rolls right and falls a

little, and then catches itself. To stop, it rolls back. The position state and the attitude state cannot move independently. They are in continuous negotiation, every 400 Hz, for the life of the flight.

**This is the same structure as turning a bike.** The kid in §1.0 knew, from years of riding, that you do not turn a bicycle by twisting the handlebars. You turn it by leaning. The lean produces the centripetal force; the handlebars are just a small auxiliary correction. A bike cannot turn without leaning. A quadcopter cannot translate without rotating. The two are structurally the same statement: *the position-state cannot move on its own; the attitude-state has to move first, and the position-state follows.* Every kid who has ridden a bicycle has felt this. The math the rest of this chapter writes is the formal version of what your body already knows.

This is what the chapter formalizes. The position state of the quadcopter is a section of one bundle. The attitude state is a section of another bundle. The two bundles are coupled by gravity and by the body-frame-to-world-frame rotation, and the coupling is the negotiation Lagrange could not yet have phrased but could already write down algebraically: *position demands an acceleration; attitude executes a reorientation; the rotation between the two reference frames mediates the conversation.*

By the end of the chapter you will see why the standard "cascaded" quadcopter controller is, structurally, a chain of bundle morphisms with explicit timescale separation. You will see why curvature in the coupling map between the two bundles is where the controller's gain schedule has to live—and why ICARUS, in Chapter 20, gates that gain schedule on a curvature obstruction rather than on hand-tuned constants. And you will see the verbatim benchmark statement from the ICARUS paper, in §5.8, because the quadcopter we are about to express as two coupled bundles is also the platform on which the framework's engineering

claims are tested.

§ 5.1

## *The position bundle*

Fix a time interval $[0, T]$ and a fixed inertial frame in $\mathbb{R}^3$ (the "world frame"). The *position bundle* of the quadcopter is the trivial fiber bundle

$$\pi_{\text{pos}} : [0, T] \times (\mathbb{R}^3 \times \mathbb{R}^3) \longrightarrow [0, T],$$

with base the time interval, fiber the position-and-velocity space $\mathbb{R}^3 \times \mathbb{R}^3$, and projection forgetting the fiber factor. A section is a smooth map

$$s_{\text{pos}} : t \longmapsto \big(\mathbf{r}(t),\, \mathbf{v}(t)\big) \in \mathbb{R}^3 \times \mathbb{R}^3$$

with $\mathbf{v} = d\mathbf{r}/dt$. The kinematic constraint $\mathbf{v} = \dot{\mathbf{r}}$ reduces the section's six free coordinates per time slice to three independent ones; we keep the redundant form because the dynamics will be written as a first-order ODE on the doubled state space.

**Dynamics.** Newton's second law, in the world frame, is

$$m\,\dot{\mathbf{v}}(t) \;=\; \mathbf{F}_{\text{world}}(t),$$

where $m$ is the quadcopter's mass and $\mathbf{F}_{\text{world}}$ is the total external force in the world frame. For a quadcopter operating in still air at low speeds, $\mathbf{F}_{\text{world}}$ has two contributions: gravity, which acts in the world frame directly, and rotor thrust, which is produced in the body frame and rotated into the world frame by the attitude. We will write this coupling explicitly in §5.3.

For now, the relevant fact is that the position bundle's dynamics depend on a quantity—$\mathbf{F}_{\text{world}}$—that the position state alone does not determine. The position bundle, considered in isolation, is an underdetermined system. The information missing from the position bundle is exactly the information the attitude bundle supplies.

**What the position bundle wants.** The control problem starts with a desired trajectory $\mathbf{r}_d(t)$ in the world frame. The position controller computes a tracking-error feedback that demands a particular world-frame acceleration:

$$\mathbf{a}_{\text{desired}}(t) \;=\; \ddot{\mathbf{r}}_d(t) \;+\; \mathbf{K}_p\big(\mathbf{r}_d(t) - \mathbf{r}(t)\big) \;+\; \mathbf{K}_d\big(\dot{\mathbf{r}}_d(t) - \mathbf{v}(t)\big).$$

The gains $\mathbf{K}_p, \mathbf{K}_d$ are positive-definite matrices that turn position and velocity error into demanded acceleration. The choice of gains is what makes a controller fast or slow, aggressive or conservative; we will return to it in §5.6 and again in Chapter 20.

The position bundle has now produced a vector $\mathbf{a}_{\text{desired}}(t) \in \mathbb{R}^3$ that it wants delivered. The bundle cannot deliver it on its own. The bundle has to talk to the attitude bundle.

§ 5.2

## *The attitude bundle*

The quadcopter's body orientation is an element of SO(3) at each instant of time. By the analysis of §3.4, we track it as an element of the universal cover $S^3$ (the unit quaternions), with the short-arc convention selecting the sign.

The *attitude bundle* is the trivial product bundle

$$\pi_{\text{att}} : [0,T] \times (S^3 \times \mathbb{R}^3) \;\longrightarrow\; [0,T],$$

with base the time interval, fiber the attitude-and-angular-velocity space $S^3 \times \mathbb{R}^3$, and projection forgetting the fiber. The $S^3$ factor of the fiber carries a left-translation action by $S^3$ that gives it the structure of a principal $S^3$-bundle over a point; the $\mathbb{R}^3$ factor takes values in the associated Lie algebra $\mathfrak{su}(2) \cong \mathbb{R}^3$. A section is a smooth map

$$s_{\text{att}} : t \longmapsto \big(q(t),\, \boldsymbol{\omega}(t)\big) \in S^3 \times \mathbb{R}^3$$

where $q(t)$ is the body-to-world rotation expressed as a unit quaternion and $\boldsymbol{\omega}(t)$ is the angular velocity expressed in the *body frame*.

The Lie-algebra structure deserves one comment. $S^3$ acts on itself by left translation, $q \mapsto q_0 \cdot q$, which is the natural action that corresponds to "rotating the attitude." The Lie algebra of $S^3$, as a Lie group, is the imaginary quaternions $\mathfrak{su}(2) \cong \mathbb{R}^3$. The body-frame angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$ is naturally an element of that Lie algebra at each instant, and reads as the connection 1-form of §3.2 pulled back to the time base along the section. The attitude dynamics that follow are the parallel-transport equation on $S^3$ written in body-frame coordinates.

**A reminder visual for what a quaternion is.** Chapter 3 §3.4 introduced quaternions formally; here is the picture you should have in your head when one shows up. Every rotation in 3D space can be described by two things: an *axis* (a unit vector saying which way the rotation points) and an *angle* (how far around the axis to rotate). That is four numbers: three for the axis vector, one for the angle. A unit quaternion packages those four numbers into a single object on the unit 3-sphere $S^3 \subset \mathbb{R}^4$, using a half-angle convention:

$$q \;=\; \cos(\theta/2) \;+\; \sin(\theta/2)(n_1 i + n_2 j + n_3 k),$$

where $\theta$ is the rotation angle, $(n_1, n_2, n_3)$ is the axis as a unit vector, and $\{i, j, k\}$ are the quaternion basis elements. The scalar part $q_0 = \cos(\theta/2)$ is just a number; the vector part $(q_1, q_2, q_3) = \sin(\theta/2)(n_1, n_2, n_3)$ encodes the axis weighted by the half-angle's sine. The four components together always satisfy $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$, which is what makes $q$ a point on the 3-sphere $S^3$.

The *half-angle* is what makes $q$ and $-q$ represent the same rotation: rotating by $\theta + 2\pi$ comes back to the same physical orientation in 3D, but flips $q$ to $-q$ in 4D (because $\cos((\theta + 2\pi)/2) = \cos(\theta/2 + \pi) = -\cos(\theta/2)$). That is the topological reason for the double cover from §3.4: every rotation has two quaternion representatives, $q$ and $-q$, on opposite poles of the 3-sphere. The short-arc convention picks the one with $q_0 \geq 0$ so the controller does not bounce between them.

For everything that follows, you can carry just this picture: *a quaternion is an axis-angle rotation packed onto a 4D unit sphere, with sign chosen to keep the controller on one hemisphere.*

**Quaternion kinematics.** The relationship between the attitude $q(t)$ and the body-frame angular velocity $\boldsymbol{\omega}(t)$ is

$$\dot{q}(t) \;=\; \tfrac{1}{2}\, q(t) \otimes \boldsymbol{\omega}_{\text{pure}}(t),$$

where $\boldsymbol{\omega}_{\text{pure}} = 0 + \omega_1 i + \omega_2 j + \omega_3 k$ is the angular velocity treated as a pure imaginary quaternion and $\otimes$ is quaternion multiplication. This is the standard quaternion kinematic equation; its derivation is in any rigid-body-dynamics text (Stevens–Lewis, Markley–Crassidis), and we take it as given.

The invariant $|q(t)|^2 = q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$ is preserved by this dynamics. A numerical integration that does not respect this invariant will drift off the manifold $S^3$; a controller that does not renormalize will eventually produce non-unit quaternions and therefore non-rotations. We are careful about this in

implementation. Chapter 14 returns to the strapdown integrator question in depth.

**Euler's equations.** The angular velocity itself evolves according to the body-frame form of Newton's second law for rotations—Euler's rigid-body equations (1765)—given the inertia tensor $\mathbf{J} \in \mathbb{R}^{3\times 3}$ and the body-frame torque $\boldsymbol{\tau}_{\text{body}}$:

$$\mathbf{J}\,\dot{\boldsymbol{\omega}}(t) \;+\; \boldsymbol{\omega}(t) \times \mathbf{J}\,\boldsymbol{\omega}(t) \;=\; \boldsymbol{\tau}_{\text{body}}(t).$$

The cross-product term $\boldsymbol{\omega} \times \mathbf{J}\,\boldsymbol{\omega}$ is the gyroscopic coupling that makes rigid-body rotation nonlinear even when the torque input is small. It is also the term that vanishes about the body's principal axes and that is generically nonzero off them.

For a symmetric quadcopter with most of the mass concentrated at the rotor arms, the inertia tensor in body principal coordinates is approximately $\mathbf{J} = \operatorname{diag}(J_{xx}, J_{yy}, J_{zz})$ with $J_{xx} \approx J_{yy}$ and $J_{zz} \approx J_{xx} + J_{yy}$ (yaw inertia roughly twice the roll/pitch inertia, by the perpendicular-axis argument applied to the four point-masses). The gyroscopic coupling $\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}$ is therefore nontrivial about all three axes; the roll/pitch asymmetry with yaw is what makes aggressive roll and pitch maneuvers couple into yaw, and why the attitude controller has to include cross-axis feedback for tightly tracked trajectories.

**Body-frame torque from motor commands.** The body-frame torque is produced by differential thrusts among the four rotors. For a standard "X" quadcopter configuration with rotors at body-frame positions $\pm\ell\hat{x} \pm \ell\hat{y}$ (so $\ell$ is the moment arm to each axis; the diagonal distance from the center to any rotor is $\sqrt{2}\,\ell$) and thrust coefficients $T_i$ for $i \in \{1, 2, 3, 4\}$,

$$\boldsymbol{\tau}_{\text{body}} \;=\; \begin{pmatrix} \ell\,(T_2 + T_3 - T_1 - T_4) \\ \ell\,(T_3 + T_4 - T_1 - T_2) \\ \kappa\,(T_1 - T_2 + T_3 - T_4) \end{pmatrix}$$

where $\kappa$ is the torque-to-thrust ratio of an individual rotor (determined by blade pitch and rotor-disk geometry). The roll torque is produced by differential thrust between the left and right pairs; the pitch torque between front and back; the yaw torque by the reaction torque of the rotors, with diagonally opposite rotors spinning in opposite directions so the net reaction sums to a controllable scalar.

The total thrust magnitude is

$$T_{\text{total}} \;=\; T_1 + T_2 + T_3 + T_4$$

which is the body-frame thrust along $\hat{z}_b$. This is the only force the rotors produce in the body frame. The body-frame force vector is $\mathbf{F}_{\text{body}} = (0, 0, T_{\text{total}})$.

That fact—that thrust has only one direction in the body frame—is the entire reason this chapter exists.

§ 5.3

## *Thrust is the bridge*

The position bundle wants a world-frame acceleration $\mathbf{a}_{\text{desired}}$. The attitude bundle has a body-frame force vector $\mathbf{F}_{\text{body}} = (0, 0, T_{\text{total}})$. The bridge between them is the body-to-world rotation matrix $\mathbf{R}(q)$ determined by the attitude quaternion $q$.

For a unit quaternion $q = q_0 + q_1 i + q_2 j + q_3 k$, the rotation matrix is

$$\mathbf{R}(q) \;=\; \begin{pmatrix} 1 - 2(q_2^2 + q_3^2) & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & 1 - 2(q_1^2 + q_2^2) \end{pmatrix}.$$

This is the standard quaternion-to-rotation-matrix formula.

Three properties matter for us:

- $\mathbf{R}(q) \in \mathrm{SO}(3)$ for every unit $q$ (the columns are orthonormal with determinant $+1$).
- $\mathbf{R}(-q) = \mathbf{R}(q)$: the sign ambiguity of the lift is invisible to $\mathbf{R}$, which is exactly the double-cover property of §3.4.
- The third column of $\mathbf{R}(q)$ is the body $\hat{z}_b$ axis expressed in the world frame, which is what carries the thrust direction.

**The world-frame thrust force.** Combining the position bundle's dynamics and the attitude bundle's force vector:

$$m\,\dot{\mathbf{v}}(t) \;=\; \mathbf{R}\big(q(t)\big)\,\mathbf{F}_{\text{body}}(t) \;+\; m\,\mathbf{g}_{\text{world}},$$

where $\mathbf{g}_{\text{world}} = (0, 0, -g)$ is the gravitational acceleration in the world frame (with $\hat{z}$ pointing up) and $\mathbf{F}_{\text{body}} = (0, 0, T_{\text{total}})$ is the body-frame thrust.

Writing $\hat{z}_b^{\text{world}}(q)$ for the third column of $\mathbf{R}(q)$—the body $Z$ axis as it appears in the world frame—this becomes

$$m\,\dot{\mathbf{v}}(t) \;=\; T_{\text{total}}(t)\,\hat{z}_b^{\text{world}}\big(q(t)\big) \;+\; m\,\mathbf{g}_{\text{world}}.$$

The position bundle's acceleration depends linearly on the thrust magnitude $T_{\text{total}}$ and on the orientation of $\hat{z}_b^{\text{world}}$, which is a nonlinear function of the attitude $q$. This is the coupling. This is the negotiation made explicit.

§ 5.4

## *Gravity is the other coupling*

Gravity acts in the world frame. The quadcopter cannot reach into the attitude bundle and turn gravity off. The attitude bundle cannot reach into the position bundle and reorient gravity. The two bundles are coupled, in a second sense, by an external force that lives in one of them and constrains the other.

This is operationally important. To hover, the quadcopter must produce a world-frame thrust force of magnitude $mg$ pointing straight up. The body $Z$ axis must therefore be aligned with the world $\hat{z}$ axis. This singles out one preferred attitude—"upright"—as the gravity-compensating equilibrium. Every other attitude requires the controller to either accept downward motion (if the thrust component along world $\hat{z}$ is less than $mg$) or to be in a transient maneuver in which downward motion is bounded.

The attitude $\theta = \pi$ (inverted, body $Z$ pointing straight down) is the most extreme case. At $\theta = \pi$, the body-frame thrust points downward in the world frame, adding to gravity rather than canceling it. The vehicle accelerates downward at $g + T_{\text{total}}/m$ until it can rotate to a less-inverted attitude. This is the InvertedRecovery benchmark scenario, which Chapter 20 reports: ICARUS recovers cleanly with 12.2× less control effort than the fixed-gain baseline, because ICARUS's holonomy-gated controller reduces effort exactly when the attitude bundle is far from gravity-aligned, and the controller has not yet earned the right to demand large torques.

This is the second face of the coupling. Thrust direction couples position to attitude. Gravity couples position back to attitude, by privileging one attitude over all others. Both are operational; both are load-bearing for the control problem.

§ 5.5

## *The algebraic negotiation*

We can now write the negotiation explicitly.

The position bundle has produced a demanded world-frame acceleration $\mathbf{a}_{\text{desired}}(t)$. From Newton's second law including gravity, the demanded thrust force in the world frame is

$$\mathbf{F}^{\text{world}}_{\text{thrust}}(t) \;=\; m\,\mathbf{a}_{\text{desired}}(t) - m\,\mathbf{g}_{\text{world}} \;=\; m\big(\mathbf{a}_{\text{desired}}(t) + g\,\hat{z}_{\text{world}}\big),$$

since $-\mathbf{g}_{\text{world}} = +g\,\hat{z}_{\text{world}}$. The demanded thrust must have magnitude

$$T_{\text{desired}}(t) \;=\; m\,\big\|\mathbf{a}_{\text{desired}}(t) + g\,\hat{z}_{\text{world}}\big\|$$

and direction

$$\hat{z}_b^{\text{desired}}(t) \;=\; \frac{\mathbf{a}_{\text{desired}}(t) + g\,\hat{z}_{\text{world}}}{\big\|\mathbf{a}_{\text{desired}}(t) + g\,\hat{z}_{\text{world}}\big\|}.$$

These two quantities together are what the position bundle demands of the attitude bundle:

$$\boxed{\begin{aligned} \mathbf{v}(t) &= \mathbf{a}_{\text{desired}}(t) + g\,\hat{z}_{\text{world}} \\ T_{\text{desired}}(t) &= m\,\big\|\mathbf{v}(t)\big\| \\ \hat{z}_b^{\text{desired}}(t) &= \frac{\mathbf{v}(t)}{\big\|\mathbf{v}(t)\big\|} \end{aligned}}$$

This is the algebraic negotiation. The position bundle sends two pieces of information to the attitude bundle: a scalar (how much thrust) and a unit vector in the world frame (which direction the body $Z$ axis should point). The attitude bundle's job is to rotate the body so that the actual $\hat{z}_b^{\text{world}}(q(t))$ matches the demanded $\hat{z}_b^{\text{desired}}(t)$, and to command the rotor speeds so that the total

thrust matches $T_{\text{desired}}(t)$.

The negotiation has a structural feature worth naming: it underdetermines the attitude. The demanded $\hat{z}_b^{\text{desired}}$ specifies only the body $Z$ direction. The body's rotation about the $Z$ axis (the yaw) is free, and is set by a separate yaw-tracking objective. Every quadcopter controller therefore has at least two attitude-level objectives: align body $Z$ with the position bundle's demand, and align the body's yaw with a separate reference (often the heading toward the next waypoint). The two objectives commute as long as the body $Z$ alignment is achieved; they conflict only in degenerate cases (e.g. at the singular configuration where the demanded $\hat{z}_b$ is itself along the yaw axis). The clean example of the degenerate case is hover: when $\hat{z}_b^{\text{desired}} = \hat{z}_{\text{world}}$, every yaw rotation produces the same body $Z$ direction, so yaw becomes a free gauge degree of freedom on the attitude bundle and the controller's yaw choice is a gauge fixing. This is one of the small places the book's gauge-symmetry theme shows up operationally.

**The negotiation can fail.** If the demanded $\hat{z}_b^{\text{desired}}$ at time $t$ is far from the actual $\hat{z}_b^{\text{world}}(q(t))$, the attitude controller cannot rotate the body fast enough to match it before the position bundle has demanded a new direction. This is exactly the failure I watched in March: the position bundle was demanding a horizontal acceleration during a full 360° roll, and the attitude bundle could not deliver the demanded $\hat{z}_b$ until the roll was past the inverted configuration. The result was a meter of lateral drift.

The negotiation has a bandwidth, and the bandwidth has a cost.

§ 5.6

## *The cascade as a chain of bundle morphisms*

Standard quadcopter control architecture is *cascaded*: outer position loop, middle attitude loop, inner angular-rate loop, with motor allocation at the bottom. Each loop runs at a different rate; the inner loops run faster than the outer loops by a deliberate timescale separation.

In bundle language, the cascade is a chain of morphisms.

1. **Position bundle → thrust-and-direction.** The position controller is a map from the position-bundle section (the trajectory so far) to a pair $(T_{\text{desired}}, \hat{z}_b^{\text{desired}})$. The map is the PD-feedback formula of §5.1 composed with the algebraic-negotiation transformation of §5.5. Loop rate: typically 50–100 Hz.

2. **Thrust-and-direction → attitude-bundle section.** The pair $(T_{\text{desired}}, \hat{z}_b^{\text{desired}})$ together with the yaw-reference $\psi_{\text{desired}}$ specifies a complete demanded attitude $q_{\text{desired}} \in S^3$. The attitude controller is a map from the attitude-bundle section (the current quaternion error $q_e = q_{\text{desired}} \otimes q^{-1}$, with short-arc sign correction) to a demanded body-frame angular velocity $\boldsymbol{\omega}_{\text{desired}}$. Loop rate: typically 200–400 Hz.

3. **Demanded angular velocity → body torque.** The angular-rate controller is a map from the angular-velocity error $\boldsymbol{\omega}_e = \boldsymbol{\omega}_{\text{desired}} - \boldsymbol{\omega}$ to a demanded body-frame torque $\boldsymbol{\tau}_{\text{body}}$. Loop rate: typically 400 Hz to 1 kHz.

4. **Body torque and total thrust → motor commands.** The motor-allocation map, called the *mixer*, inverts the

linear map $(T_1, T_2, T_3, T_4) \mapsto (T_{\text{total}}, \boldsymbol{\tau}_{\text{body}})$ derived in §5.2. For a standard "X" configuration this is a $4 \times 4$ matrix inversion, computed once and stored. The mixer matrix is invertible generically but loses rank near actuator saturation: when one or more motors hit their positive or negative thrust limits, the linear allocation map can no longer deliver an arbitrary $(T_{\text{total}}, \boldsymbol{\tau}_{\text{body}})$ pair, and a saturation-handling layer (e.g. desaturation by prioritizing attitude torque over total thrust) has to be added on top of the linear mixer. Saturation is exactly what happens during the high-torque phases of MultirotorFlip and InvertedRecovery, and any honest implementation has to address it.

Each step is a bundle morphism: a smooth map from one bundle's sections (or from a partial-state of one bundle) to another's. The composition runs from world-frame trajectory desire at the top to four motor PWM signals at the bottom. The composition is what makes the quadcopter fly.

**Why the cascade works (when it works).** The cascade depends on the inner loops being faster than the outer loops by enough margin that the inner loops have effectively converged before the outer loop has demanded a new reference. If the angular-rate loop runs at 400 Hz and the position loop runs at 50 Hz, the angular-rate loop has eight cycles per position-loop step—roughly enough to settle a first-order error response to within a few percent before the position loop demands a new $\boldsymbol{\omega}_{\text{desired}}$. Timescale separation is the architectural commitment that makes the cascaded controller stable.

It is also the architectural commitment that breaks when the maneuver is aggressive enough to demand angular velocities the inner loops cannot deliver in one outer-loop step. This is exactly the MultirotorFlip failure mode of §5.0. The cascade does not fail because of bad gains. The cascade fails because the timescale

separation it requires has been violated by the maneuver itself.

ICARUS's response to this is structural rather than tuning: it gates the outer loop's gain on a curvature obstruction derived from the predicted trajectory, so that the outer loop demands less aggressive accelerations exactly when the inner loops are most likely to fall behind. §5.7 takes up the curvature, and Chapter 20 reports the result.

§ 5.7

## *Where the curvature lives*

We have a coupled system. The position bundle has dynamics that depend on the attitude bundle's section. The attitude bundle has dynamics that depend on the body torque, which is produced by motor commands that the cascade computes from the position bundle's demands. The composition is a closed loop.

The natural place for curvature, in the sense of §3.2 and §3.3, is the coupling map between the two bundles—specifically, the map

$$q \;\longmapsto\; \hat{z}_b^{\text{world}}(q) \;=\; \text{third column of } \mathbf{R}(q).$$

This map sends a quaternion in $S^3$ to a unit vector in $\mathbb{R}^3$. Its image is the unit sphere $S^2 \subset \mathbb{R}^3$. The map factors through the standard Hopf-like fibration $S^3 \to S^2$, with fiber $S^1$ corresponding to the unconstrained yaw freedom[1].

[1]The strict Hopf fibration $S^3 \to S^2$ is the map $q \mapsto q \cdot \mathbf{k} \cdot q^*$, sending a unit quaternion to the image of the basis quaternion $\mathbf{k}$ under conjugation, which is the second column of $\mathbf{R}(q)$. Our coupling map uses the third column ($\hat{z}_b^{\text{world}}$), which differs from the Hopf map by the choice of which body axis is the thrust direction. The two maps are related by a fixed reordering of basis quaternions; the topological content—non-trivial $S^1$-bundle over $S^2$ with total space $S^3$—is the same. "Hopf-like" is honest hedging on the convention, not on the topology.

The differential

$$\left.\frac{\partial \hat{z}_b^{\text{world}}}{\partial q}\right|_q : T_q S^3 \longrightarrow T_{\hat{z}_b(q)} S^2$$

varies smoothly with $q$ and is surjective everywhere (the Hopf fibration is a submersion). Pointwise, the coupling map's differential does not degenerate. What changes between hover and inversion is *not* the rank of the differential, but a cluster of operational obstructions that all peak together along high-amplitude attitude trajectories:

- *Sign discontinuity in naive subtraction-based attitude error.* The double cover $S^3 \to \mathrm{SO}(3)$ identifies $q$ and $-q$. A controller that computes $q_{\text{cmd}} - q$ in $\mathbb{R}^4$ hits a sign flip when the trajectory passes near $\theta = \pi$, and the wrong sign of the error vector commands the long way around the $S^3$ great circle. The standard fix is the multiplicative attitude error $q_{\text{err}} = q_{\text{cmd}}^{-1} \otimes q$ with short-arc sign correction; with that fix in place the error itself is well-defined at $\theta = \pi$, but the next two obstructions remain.

- *Cascade-bandwidth deficit.* The outer position loop runs at 50–100 Hz; the inner attitude loop, at 400–1000 Hz. The outer loop's $\hat{z}_b^{\text{desired}}$ can swing through large angles in a few outer-loop ticks; the inner loop has finitely many inner-loop ticks to slew the body to match. The mismatch is a timescale-separation violation, not a differential-geometric degeneracy.

- *Thrust-feasibility and motor saturation.* Rotor thrust is non-negative and aligned with $+\hat{z}_b$. Near inversion the body thrust direction is opposed to gravity, so the controller must slew the body before it can produce a corrective acceleration; meanwhile the torque commands needed for an aggressive slew saturate the motors.

These are the obstructions a curvature-aware controller has to respect along trajectories that approach inversion. The framework's reading is that they all show up as elevated $\hat{\kappa}$ along the predicted trajectory through the coupling, not as a rank loss in the Hopf-like map's differential. The $\mathbb{Z}/2$ residue of $\pi_1(\mathrm{SO}(3))$ from §3.4 is what *forces* the sign-correction obstruction; the bandwidth and saturation obstructions are physical, not topological. The curvature obstruction $\hat{\kappa}$ is the operational scalar the framework uses to fold all three into a single gating signal.

The Davis-Geometric curvature obstruction $\hat{\kappa}$ on this coupling map measures, dimensionlessly and non-negatively, exactly how much of this degeneracy the controller is about to face along its predicted trajectory. The operational confidence $C = 1/(1 + \hat{\kappa})$ gates the controller's gain accordingly: high $C$ (low $\hat{\kappa}$) means the predicted trajectory stays in the flat region near hover, and the controller can run at full gain; low $C$ (high $\hat{\kappa}$) means the predicted trajectory crosses the degenerate region near inversion, and the controller's gain is throttled to give the attitude inner loop time to catch up.

This is what "holonomy-gated control" means on the quadcopter. The gating is on a real curvature obstruction in a real coupling map between two real fiber bundles, computed in real time against the trajectory the controller is about to execute. It is not gain-scheduled on hand-picked breakpoints. It is gated on a geometric invariant of the coupling.

Chapter 20 reports what this gating accomplishes on the benchmark suite. §5.8 gives the headline.

§ 5.8

## *Forward to Chapter 20: the benchmark*

*The remainder of this section is a forward pointer to the benchmark results developed in full in Chapter 20. It is included here so that the quadcopter we have just expressed as two coupled bundles is, before the chapter closes, named as the platform on which the framework's engineering claims are tested.*

The quadcopter we just expressed as two fiber bundles is the platform that ICARUS uses for its benchmark suite. ICARUS—the flight control architecture that emerged from this geometric reading of flight—is benchmarked against a fixed-gain quaternion LQR baseline (with a separate inversion controller for full-attitude maneuvers) on four scenarios: HoverStability (baseline check), MultirotorFlip (360° roll), InvertedRecovery (initial attitude $\theta = \pi$), and SwingUpSinglePendulum (passage through unstable equilibrium).

On the three scenarios validated in the paper to date, ICARUS achieves 100% success across $n = 500$ Van der Corput-perturbed **quasi-Monte Carlo** trials per scenario (see Chapter 20, §23.5, for the full benchmark table and binomial-reference intervals). On MultirotorFlip, the fixed-gain baseline (without short-arc sign correction) crashes at $\theta \approx \pi$ in 100% of trials. On InvertedRecovery, both controllers succeed but ICARUS uses 12.2× less control effort and produces 3.2× lower integrated squared angular acceleration. On HoverStability, ICARUS uses 22% less control effort, confirming the absence of regression on easy cases. SwingUpSinglePendulum is not yet validated.

Chapter 20 also reports the **modern fixed-gain geometric baseline comparison** (BaselineGeom, Lee–Leok–McClamroch class with short-arc correction): ICARUS wins on InvertedRecovery (12.07× effort reduction) and HoverStability (1.28× effort

reduction) but loses on MultirotorFlip (2.4× *more* effort)—a regime-specific result, not universal dominance. The full benchmark table, ablation study, and binomial-reference intervals are in Chapter 20 and Appendix E.

The result is regime-specific by construction. ICARUS is built to win on recovery scenarios (high curvature in the coupling, high $\hat{\kappa}$, low $C$, throttled gain that lets the inner loop catch up). It is built to be neutral or slightly better on easy cases (low curvature, high $C$, gains close to baseline). It is *not* built to win on commanded fast slews (low curvature in the coupling but high commanded angular acceleration), because in those scenarios the gain throttle costs the controller bandwidth it does not need to give up. The framework predicts the regime structure of the result before the benchmark is run. The benchmark is the falsification test.

This is what I mean when I say the engineering thesis of the book is falsifiable: the framework makes a specific prediction about *when* ICARUS will beat the baseline and *when* it will lose, and the benchmark either confirms the prediction or it does not. Chapter 20 reports that it does.

§ 5.9

## *What you take from this chapter*

Three things.

**First.** A quadcopter's state lives on two coupled fiber bundles. The position bundle (translational state) and the attitude bundle (rotational state) are coupled by gravity and by the body-frame-to-world-frame rotation. The coupling is *algebraic* in Lagrange's sense: the bundles negotiate through equations, not through any

internal mechanical link, and the controller's job is to mediate the negotiation.

**Second.** The standard cascaded quadcopter controller is a chain of bundle morphisms. Position bundle → thrust-and-direction → attitude-bundle section → angular velocity → body torque → motor commands. Each morphism has a loop rate, a gain, and a timescale-separation requirement. When the maneuver violates the timescale separation, the cascade fails—not from bad gains, but from a structural mismatch between the demand and the bandwidth.

**Third.** The curvature obstruction $\hat{\kappa}$ that gates the controller's gain schedule does *not* measure rank loss in the coupling map's differential—the Hopf-like map is a submersion everywhere. It measures the combined operational obstruction (short-arc sign correction, cascade-bandwidth deficit, thrust feasibility, motor saturation) along the predicted trajectory through the coupling. Near hover, all four are slack and full gain is safe. Far from hover, especially near inversion, they all tighten together and gain must be throttled. ICARUS computes this throttle in real time from a curvature obstruction $\hat{\kappa}$, rather than from hand-tuned breakpoints. The benchmark result of §5.8 is the operational consequence.

Chapter 6 takes the same machinery to a richer plant: the F-16 longitudinal model. There the body bundle is more complex (aerodynamic forces depend nonlinearly on velocity and attitude through the angle of attack), the gauge group is larger (the model has more symmetries than the equations of motion make visible), and the curvature in the coupling lives in places the quadcopter's coupling does not have. By the end of Chapter 6, the geometric reading of a state-space model will be in place, and Part III—frequency, optimal, robust—will be ready to start.

*Vamos.*

Chapter 6

# *The F-16 longitudinal model and its gauge symmetries*

*Symmetry, as wide or as narrow as you may define its meaning, is one idea by which man through the ages has tried to comprehend and create order, beauty, and perfection.*
*Hermann Weyl, Symmetry (1952)*

§ 6.0

## *Why the F-16 is in this book*

There is a state-space model of the F-16 fighter at the front of every aerospace control textbook in print. The version most readers have seen is in Stevens and Lewis, *Aircraft Control and Simulation*—a textbook that has gone through three editions, that I have owned a copy of since 1998, and that every American GNC engineer of my generation has worked through at least once. The trim condition is Mach 0.6 at 15,000 feet, level flight, mid-weight configuration. The matrices are $4 \times 4$ for the longitudinal subsystem, $5 \times 5$ for the lateral. The eigenvalues are well-behaved.

The model has been the entrance examination for fifty years.

I am putting it in this book for one specific reason, and I want to be honest about the reason before the chapter starts. The F-16 longitudinal model is the standard worked example in the field. It is the example every control theorist knows, every aerospace engineer has computed eigenvalues for, every textbook reproduces. If the geometric reading of classical control this book is making is correct, it had better hold up on the worked example. So I am going to take Stevens–Lewis's F-16 and read it geometrically. I will name the gauge symmetries, identify what about the matrix representation is real and what is a coordinate convention, and show how the controller bundle of Chapter 5 specializes when the plant is fixed-wing rather than multirotor.

The chapter is shorter and more disciplined than the chapters that came before. I am not going to repeat the bird-and-self autobiography here; I have asked you to grant it. What I am going to ask you to grant in this chapter is that you have a Stevens–Lewis open on your desk or that you can pull up the numerical A and B matrices when you need them. I will reproduce the structure and the symbolic stability derivatives that the geometric reading depends on. The exact numerical entries (at the M0.6, h = 15,000 ft trim) are in Stevens–Lewis Table 3.4-1 in the third edition; if you want to run the matrices through MATLAB or Python at the end of the chapter and reproduce my eigenvalue calculations, that is the table to look at.

The chapter has six things in it.

1. The longitudinal subsystem, formally: states, controls, trim. (§6.1)

2. The Stevens–Lewis A and B matrices at the canonical trim, in symbolic stability-derivative form. (§6.2)

3. The claim that the linearization is a gauge choice, and what

that means precisely. (§6.3)

4. The gauge group of the F-16 longitudinal model, listed and named. (§6.4)

5. The F-16 model as a section of the controller bundle of Chapter 5, specialized to fixed-wing. (§6.5)

6. Which control-design quantities are gauge-invariant and which are not, with operational consequences. (§6.6)

By the end of the chapter, the geometric reading of classical control will be in place on a non-toy plant. Part III will then take that reading and run it through Bode, LQR, and $H_\infty$.

§ 6.1

## *The longitudinal subsystem and the trim point*

The longitudinal-lateral decomposition of fixed-wing aircraft dynamics is the first analytical simplification every textbook makes. The full six-degree-of-freedom rigid-body model is twelve-dimensional (position $\mathbf{r} \in \mathbb{R}^3$, velocity $\mathbf{v} \in \mathbb{R}^3$, attitude $q \in S^3$, angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$), plus actuator dynamics. The longitudinal subsystem is the four-dimensional restriction obtained by assuming the lateral states (sideslip $\beta$, roll rate $p$, yaw rate $r$, roll angle $\phi$, heading $\psi$) are decoupled from the longitudinal dynamics and held at zero.

The longitudinal state vector is

$$\mathbf{x}_{\text{lon}} \;=\; \begin{pmatrix} V \\ \alpha \\ q \\ \theta \end{pmatrix},$$

where $V$ is the total airspeed (ft/s), $\alpha$ is the angle of attack (rad), $q$ is the body-frame pitch rate (rad/s), and $\theta$ is the body pitch angle relative to the world horizon (rad). The control input, for the trimmed-throttle case, is the elevator deflection

$$u \;=\; \delta_e$$

in radians (positive elevator-down conventionally producing nose-down pitching moment). With throttle considered a separate input, the control vector is $\mathbf{u} = (\delta_e, \delta_T)^{\mathrm{T}}$; for most of this chapter I will hold throttle at trim and consider $\delta_e$ alone.

**The trim point.** The standard worked trim in Stevens–Lewis is Mach 0.6 at 15,000 ft pressure altitude, mid-weight configuration. At this trim:

- Airspeed $V_0 \approx 597$ ft/s (Mach 0.6 at the standard-atmosphere temperature at 15,000 ft).
- Trim angle of attack $\alpha_0 \approx 0.039$ rad ($\approx 2.2^\circ$), set so that lift equals weight at $V_0$.
- Trim pitch rate $q_0 = 0$ (level flight).
- Trim pitch angle $\theta_0 = \alpha_0 \approx 0.039$ rad (level flight, so the flight-path angle is zero, and pitch equals angle of attack).
- Trim elevator $\delta_{e,0}$ approximately $-0.02$ rad ($-1.1^\circ$), set to produce zero pitching moment at the trim.
- Trim throttle $\delta_{T,0}$ set to produce thrust equal to drag at the trim.

The trim is the equilibrium of the full nonlinear dynamics with $\dot{\mathbf{x}} = 0$ in the longitudinal states, achieved at the specified airspeed and altitude by adjusting elevator and throttle until lift balances weight, thrust balances drag, and pitching moment vanishes.

**The linearization.** Around the trim point, the nonlinear equations of motion are expanded to first order in the perturbation states $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_0$ and perturbation control $\Delta\mathbf{u} = \mathbf{u} - \mathbf{u}_0$. The resulting linear time-invariant system is

$$\Delta\dot{\mathbf{x}}_{\text{lon}} \;=\; \mathbf{A}\,\Delta\mathbf{x}_{\text{lon}} \;+\; \mathbf{B}\,\Delta\mathbf{u},$$

where $\mathbf{A} \in \mathbb{R}^{4\times 4}$ and $\mathbf{B} \in \mathbb{R}^{4\times 1}$ (or $4 \times 2$ if throttle is included). The entries of $\mathbf{A}$ and $\mathbf{B}$ are the standard stability and control derivatives evaluated at the trim. The next section writes them.

§ 6.2

## *The A and B matrices, in symbolic form*

The longitudinal A matrix at the canonical trim, written in the symbolic stability-derivative notation that every aerospace textbook uses[1]:

$$\mathbf{A} \;=\; \begin{pmatrix} X_V & X_\alpha & 0 & -g\cos\theta_0 \\ Z_V/V_0 & Z_\alpha/V_0 & 1 + Z_q/V_0 & -g\sin\theta_0/V_0 \\ M_V & M_\alpha & M_q & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

The notation is standard but worth unpacking once:

- $X_V, X_\alpha$: longitudinal (forward-axis) force derivatives with respect to airspeed and angle of attack.

[1]The form shown here suppresses the unsteady-aerodynamic coupling terms proportional to $M_{\dot\alpha}$, which appear as additive corrections in the third row of the full Stevens–Lewis matrix. The $M_{\dot\alpha}$ contribution is small at the canonical trim and does not affect the eigenvalue structure qualitatively; I am leaving it out so the matrix fits on the page. Stevens–Lewis (3rd ed.) Equation 3.4-1 gives the full form.

- $Z_V, Z_\alpha, Z_q$: vertical (downward-axis) force derivatives.
- $M_V, M_\alpha, M_q, M_{\dot\alpha}$: pitching-moment derivatives, including the $\dot\alpha$ term that captures unsteady-aerodynamic effects.
- $g$: gravitational acceleration.
- $\theta_0$: the trim pitch angle.

The B matrix for elevator-only control is

$$\mathbf{B}_{\delta_e} = \begin{pmatrix} X_{\delta_e} \\ Z_{\delta_e}/V_0 \\ M_{\delta_e} + M_{\dot\alpha}\, Z_{\delta_e}/V_0 \\ 0 \end{pmatrix},$$

where $X_{\delta_e}, Z_{\delta_e}, M_{\delta_e}$ are the force and moment derivatives with respect to elevator deflection.

**Numerical values.** The numerical values of these derivatives at the M0.6, h = 15,000 ft trim are in Stevens–Lewis (3rd edition) Table 3.4-1. I am not reproducing the table here—partly because it is straightforward to look up, partly because the numerical values are not what this chapter is about. What this chapter is about is the structure of the matrices and what is gauge-invariant in it.

For reference, however: the eigenvalues of $\mathbf{A}$ at this trim partition into two complex conjugate pairs. One pair is the *short-period* mode (rapid pitch oscillation, frequency $\approx 2$ rad/s, damping ratio $\approx 0.6$). The other pair is the *phugoid* mode (slow exchange of altitude and airspeed, frequency $\approx 0.1$ rad/s, damping ratio $\approx 0.05$). These two modal labels are the canonical longitudinal-dynamics decomposition; they appear in every textbook and they are the practical operational targets that any longitudinal control law must shape.

The B matrix's third entry (the elevator-to-pitch-rate gain, $M_{\delta_e}$) is the dominant control authority: elevator deflection produces pitch acceleration directly, and the other control couplings (elevator-to-axial-force, elevator-to-vertical-force) are smaller corrections.

Now the chapter's actual work begins.

§ 6.3

## *The linearization is a gauge choice*

The A and B matrices I just wrote depend on choices that are not part of the underlying physics. Three of them, in particular, are worth pulling out explicitly because they will reappear when we name the gauge group in §6.4.

**Choice 1: the trim point.** The A and B matrices are evaluated at the specific trim $\mathbf{x}_0 = (V_0, \alpha_0, 0, \theta_0)$, $\mathbf{u}_0 = (\delta_{e,0}, \delta_{T,0})$. A different trim—say, M = 0.8 at 30,000 ft, or M = 0.4 at sea level—gives different numerical values for the stability derivatives. The matrices in the textbook are exactly the M = 0.6, h = 15,000 ft instance of a family of linearizations parameterized by the trim manifold. The physics is the same; the matrix entries are the trim-dependent local presentation of that physics.

**Choice 2: the body-axis frame.** The angle of attack $\alpha$ is the angle between the body's $X$-axis and the freestream velocity vector, projected onto the body's symmetry plane. The body $X$-axis is conventionally taken along the aircraft's longitudinal axis. But "the longitudinal axis" is itself a choice. Three conventions are in common use:

- *Body-fixed axes*: $X_b$ along the fuselage reference line.

- *Stability axes*: $X_s$ along the freestream velocity (rotated from body-fixed by $\alpha_0$).
- *Wind axes*: $X_w$ along the relative wind, accounting for sideslip (degenerate for purely longitudinal motion).

The stability derivatives transform between these axes under explicit rotation rules. Different textbooks use different conventions; Stevens–Lewis defaults to body-fixed, while Roskam and others default to stability axes. The transformation is invertible and smooth—the underlying physics is gauge-invariant; the matrix representation is gauge-dependent on which axes you chose.

**Choice 3: the units.** The matrices I wrote in §6.2 have airspeed in ft/s, angles in radians, and pitching moment in foot-pounds. The same physics in SI units—m/s, radians, Newton-meters—gives numerically different matrix entries, related by a diagonal scaling transformation. Some textbooks nondimensionalize the derivatives (e.g. $C_{L_\alpha}$ instead of $Z_\alpha$), absorbing the dynamic pressure and reference area into the coefficient. The nondimensional form is gauge-fixed in a different way than the dimensional form, and the two forms transform into each other under an explicit (but tedious) rescaling.

**What gauge-dependence means.** The numerical entries of $\mathbf{A}$ and $\mathbf{B}$ are not the natural object. The natural object is the dynamical system itself—the smooth flow on the longitudinal state manifold, restricted to a neighborhood of the trim. The matrices are a coordinate-dependent presentation of that flow, in a specific axis frame, at a specific trim, in specific units.

Two engineers presented with the M0.6, h = 15,000 ft trim in stability axes vs body-fixed axes will write down different numerical entries for $\mathbf{A}$ but the *same* dynamics. Their eigenvalues will match. Their step responses will match. Their controller designs, expressed in their respective coordinate systems, will

produce the same closed-loop behavior. The dynamics is gauge-invariant; the matrix entries are gauge-dependent.

This is the kind of statement that sounds either obvious ("of course coordinates are a convention") or radical ("the equations of motion change under transformation") depending on the reader's background. I want to argue, in the rest of this chapter, that it is the right level of statement: precise enough to be useful, general enough to be transferable to the rest of the book.

§ 6.4

## *The gauge group of the longitudinal model*

Let me name the gauge group explicitly.

The longitudinal F-16 model has a Lie group of symmetries that act on the matrix representation while leaving the physics invariant. The group is a product of several factors, each capturing one of the choices from §6.3. I will list them.

1. **Spatial translations in the world frame.** The dynamics are invariant under shifts of the world coordinate origin. Three independent translations: $\mathbb{R}^3$. The matrices $\mathbf{A}, \mathbf{B}$ are insensitive to this factor because position is not a state in the longitudinal subsystem (the longitudinal states are velocity, angle of attack, pitch rate, pitch angle—all coordinate-independent). The translation symmetry is the physical statement that flight at 15,000 ft over Albuquerque has the same longitudinal dynamics as flight at 15,000 ft over Indianapolis.

2. **World yaw rotation.** Rotation about the world $\hat{z}$ axis (yaw) is a one-parameter group $\mathrm{SO}(2) \cong S^1$. The longitudinal dynamics are invariant under this because longitudinal

motion is by construction in the body's symmetry plane, and rotating that plane in azimuth does not change the dynamics within it. The lateral subsystem breaks this symmetry; the longitudinal-lateral decomposition is exactly the statement that yaw is a free gauge degree of freedom for the longitudinal model.

3. **Time translation.** The dynamics are autonomous (the $\mathbf{A}, \mathbf{B}$ matrices do not depend explicitly on $t$). Time translation is a one-parameter group $\mathbb{R}$. Standard for any LTI system.

4. **Body-axis rotation about the body $\hat{y}$ axis (pitch).** This is the gauge factor most easily missed. The choice between body-fixed axes and stability axes is a rotation of the body frame about the body $\hat{y}$ axis (pitch axis), through the trim angle of attack $\alpha_0$. Stability axes are body-fixed axes rotated by $-\alpha_0$ so that the new body $X$-axis aligns with the freestream velocity. This is a one-parameter rotation group SO(2) acting on the body axes; the matrices transform under a similarity transformation $\mathbf{A} \to \mathbf{TAT}^{-1}$ with $\mathbf{T}$ the rotation, and $\mathbf{B}$ transforms covariantly. The eigenvalues are invariant; the matrix entries are not.

5. **Unit scaling.** Scaling the airspeed unit by a factor $\lambda$ (e.g. converting ft/s to m/s), the angle unit by a factor $\mu$ (e.g. radians to degrees), and the time unit by a factor $\nu$ produces a diagonal scaling on the state vector and a coupled scaling on the matrix entries. The unit group is $\mathbb{R}^3_+$ (three independent positive scalings). Nondimensionalization is a specific gauge fixing within this group: it absorbs the dynamic pressure and reference geometry into the coefficient definitions.

6. **Linearization-point shift.** Different trim conditions

give different matrices. The trim manifold is a submanifold of the state space (typically two-dimensional, parameterized by airspeed and altitude, or equivalently by Mach number and dynamic pressure). Shifts along the trim manifold are a continuous group action $\mathbb{R}^2$ that takes one A-matrix to another within the family. This is not strictly a gauge symmetry of one specific model; it is the parameterization of the family of linearizations to which any one specific model belongs.

The full gauge group, for one specific linearization, is the product

$$G_{\text{lon}} \;=\; \mathbb{R}^3 \times S^1 \times \mathbb{R} \times \mathrm{SO}(2) \times \mathbb{R}^3_+$$

(spatial translation, yaw, time, body-pitch rotation, unit scaling), with the trim-shift $\mathbb{R}^2$ being a separate "configuration-space" action that connects different specific models within the family.

$G_{\text{lon}}$ is a nine-dimensional Lie group: three dimensions from $\mathbb{R}^3$ (spatial translations), one from $S^1$ (world yaw), one from $\mathbb{R}$ (time translation), one from $\mathrm{SO}(2)$ (body-pitch axis rotation), and three from $\mathbb{R}^3_+$ (independent rescalings of airspeed, angle, and time units). The trim-shift $\mathbb{R}^2$ is not counted here because it is a configuration-space action that takes one specific linearization to another within the family, rather than a gauge symmetry of one specific model; if you include the trim-shift, the full family of linearizations sits inside an eleven-dimensional group action. The matrix entries of $\mathbf{A}, \mathbf{B}$ are coordinates on the orbit of the dynamical system under $G_{\text{lon}}$, with the matrix presentation being a particular choice of representative.

This is what "coordinate-dependent presentation" means precisely. The orbit is the gauge-invariant object. The representative is the matrix.

§ 6.5

## *The F-16 model as a section of the controller bundle*

Chapter 5 expressed the quadcopter's state as living on two coupled fiber bundles—position and attitude—with the cascade controller as a chain of bundle morphisms. The F-16 longitudinal model is the same kind of object on a different platform.

**The plant bundle.** The longitudinal state space is the 4-manifold $V \times \alpha \times q \times \theta \cong \mathbb{R}_+ \times \mathbb{R} \times \mathbb{R} \times S^1$ (airspeed is positive; pitch angle is naturally on the circle, though usually treated as $\mathbb{R}$ near trim). The trivial bundle $[0, T] \times \mathbb{R}^4 \to [0, T]$ with sections being trajectories is the analogue of the position bundle from Chapter 5.

**The controller bundle.** The controller's task is to map state trajectories to elevator and throttle commands. Formally, the controller is a section of a controller bundle whose base is the plant state and whose fiber, at each state, is the control input space. A linear controller $\mathbf{u} = -\mathbf{K}\Delta\mathbf{x}$ is a particular constant section of this bundle (the gain matrix $\mathbf{K}$ assigning a control input to every perturbation state). LQR (Chapter 8) and $H_\infty$ (Chapter 9) are different choices of constant section, optimizing different cost functions over the same plant bundle. ICARUS, when specialized to fixed-wing, would be a non-constant section: $\mathbf{K}$ modulated by the operational confidence $C = 1/(1 + \hat{\kappa})$ along the predicted trajectory.

**The gauge action on the controller.** When the gauge group of §6.4 acts on the plant matrix $(\mathbf{A}, \mathbf{B})$, it simultaneously acts on the controller gain $\mathbf{K}$ in a coordinated way. Specifically: a similarity transformation $\mathbf{A} \to \mathbf{TAT}^{-1}$, $\mathbf{B} \to \mathbf{TB}$ (corresponding to a

coordinate change in the state) induces $\mathbf{K} \to \mathbf{K}\mathbf{T}^{-1}$. The closed-loop matrix $\mathbf{A} - \mathbf{B}\mathbf{K}$ transforms as $\mathbf{A} - \mathbf{B}\mathbf{K} \to \mathbf{T}(\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{T}^{-1}$, which preserves its eigenvalues. The closed-loop dynamics are gauge-invariant; the numerical entries of $\mathbf{K}$ are gauge-dependent.

This is the geometric reading of the standard transformation rules for state-space models. The rules every textbook gives—"you must rotate $\mathbf{K}$ along with $\mathbf{A}$ when you change coordinates"—are the gauge transformation rules for the controller bundle.

**The kid's bike had gauge symmetries too.** The lawnmower-bike of §1.0 did not care which driveway it sat in (spatial translation), which day of the week it was (time translation), or which way north was (yaw rotation). The dynamics of the bike—if the kid had ever started it—would have been the same on Tuesday in Detroit as on Saturday in Houston. The bike has the same translation, time, and yaw symmetries the F-16 has. The kid's intuitive understanding that "a bike is a bike, wherever and whenever you ride it" is the gauge-invariance claim made informal. The F-16's nine-dimensional gauge group is what that informal sentence looks like when you write it down formally for a vehicle whose dynamics matter.

§ 6.6

## *Gauge-invariant vs gauge-dependent quantities*

The control-theoretic upshot of the gauge analysis is a clean distinction between what the system is really doing and what your particular matrix is reporting.

**Gauge-invariant quantities.**

- *Eigenvalues of* $\mathbf{A}$. The closed-loop and open-loop poles

do not depend on coordinate choice. Stability is gauge-invariant. Modal damping ratios and natural frequencies are gauge-invariant.

- *Transfer function poles and zeros, and the transfer function itself in input-output form.* The Bode plot of $G(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}$ is invariant under the gauge group's similarity action on $\mathbf{A}, \mathbf{B}, \mathbf{C}$. This is why Bode (Chapter 7) is the natural setting for many design specifications: it expresses the system in a gauge-invariant way.

- *Singular values of the closed-loop sensitivity and complementary sensitivity functions.* These determine the robustness margins of the closed-loop system and are gauge-invariant.

- *Controllability and observability of the system.* A coordinate transformation cannot create or destroy controllability or observability; these are gauge-invariant properties.

**Gauge-dependent quantities.**

- *Numerical entries of* $\mathbf{A}, \mathbf{B}, \mathbf{K}$. As discussed.

- *LQR gain values for a specific state-weight choice* $\mathbf{Q}$. The matrix $\mathbf{Q}$ in the LQR cost $\int (\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u})\, dt$ has units that depend on the units of the state. Changing units changes $\mathbf{Q}$, and the resulting LQR gains. The closed-loop eigenvalues (which the LQR optimization is implicitly setting) are gauge-invariant; the gain values are not.

- *The Kalman filter's covariance matrix* $\mathbf{P}$. The matrix entries depend on the units of the state. The filter's prediction (state estimate) is gauge-equivariant: it transforms under the gauge group along with the state coordinates. The trace of $\mathbf{P}$ in particular units is not directly meaningful; what is

meaningful is the Fisher-information-derived dimensionless uncertainty scalar (Chapter 10, §10).

**Operational consequence.** When you specify a design requirement, you should specify it in gauge-invariant language wherever possible. "Damp the short-period mode to $\zeta \geq 0.5$" is gauge-invariant. "Set the (1,3) entry of $\mathbf{K}$ to 0.4" is gauge-dependent and useless without specifying the gauge. When you communicate a design between teams, you should communicate in the gauge-invariant form; otherwise the receiving team will have to know your unit conventions and axis frame to make sense of your numbers.

This is not new advice. Aerospace engineers learn it early and forget it often. The geometric reading makes the advice precise: gauge-invariance is the right level of abstraction at which the design specification lives.

## § 6.7

## *What you take from this chapter*

The F-16 longitudinal model is a coordinate-dependent presentation of a coordinate-free dynamical system. The matrices in Stevens–Lewis Table 3.4-1 are one particular presentation, at one specific trim, in one specific axis frame, in one specific unit system. The gauge group acting on this presentation is at least six-dimensional, factoring into spatial translations, world yaw, time translation, body-axis rotation (the body-fixed vs stability-axes choice), unit scaling, and trim-point shift along the linearization manifold.

The eigenvalues of $\mathbf{A}$, the transfer function poles and zeros, the singular values of the closed-loop sensitivity functions, and

the system's controllability/observability are gauge-invariant. The numerical entries of $\mathbf{A}, \mathbf{B}, \mathbf{K}$, the LQR gain values for a specific $\mathbf{Q}$, and the covariance matrix entries are gauge-dependent. Design specifications should be expressed in the gauge-invariant quantities; gauge-dependent values are useful only with their full coordinate context.

The F-16 model, in the geometric reading of this book, is a section of a controller bundle. The choice of $\mathbf{A}, \mathbf{B}$ representation is a choice of trivialization of that bundle in a particular coordinate patch on the plant state manifold. The standard textbook design procedures (LQR, $H_\infty$, Bode loop-shaping) all operate on this presentation, but they do not depend on it; what they depend on is the underlying gauge-invariant dynamical content.

Part III takes this in three steps. Chapter 7 works the frequency-domain reading of the same F-16 model and shows that Bode plots are computing a gauge-invariant feature of the system. Chapter 8 does LQR and shows where the gauge-dependence shows up in the gain matrix and how to read past it. Chapter 9 does $H_\infty$ and identifies the gauge-invariant robustness margins that $H_\infty$ synthesis is shaping. The bundle structure of the F-16 is the connecting tissue.

*Vamos.*

Chapter 7

# *Bode, sensitivity, and the Curvature Floor*

*Unstable systems are fundamentally, and quantifiably, more difficult to control than stable ones.*
*Gunter Stein, "Respect the Unstable," Bode Lecture (1989); reprinted in IEEE Control Systems Magazine 23(4), August 2003*

## § 7.0

## *Where the conservation law comes from*

The kid is riding her BMX down a road with bumps in it. Some of the bumps are small and come fast—loose gravel, hairline cracks, the chatter of a textured surface. Some are big and come slow—potholes, root-lifted pavement, the lip where the driveway meets the street. The bumps shake her around. She has some control over how much they shake her: she can crouch low and let her arms absorb the impact, or stiffen up and ride hard through. She cannot eliminate the shaking; she can only redistribute it.

Crouch loose, and the big bumps slam her—her arms compress fully and the load goes straight into her shoulders. Stiffen up, and the small chatter rattles her teeth—no compliance, every pebble transmits. The choice is between which kind of bump hurts more. *The total amount of shaking, integrated over all the bump-sizes she encounters along the road, is roughly conserved.* She can shift the discomfort from low-frequency to high-frequency or back, but she cannot make it disappear.

This chapter is about that conservation law. The control-theory name for it is the **Bode sensitivity integral**. The kid was experiencing it in her arms; the integral is the formal statement of what she could not change. Before we write the integral down, the same words the integral uses, in plain English:

- *Feedback* is what the kid's body is doing automatically. Her arms sense the bump (input), her muscles produce a response (output), her body responds again to that response, and so on. A closed loop. "Feedback amplifier" is the 1945-era electrical-engineering name for the same kind of closed loop, applied to electronic signals: the amplifier reads its own output, compares it to the desired output, and uses the difference to drive the next moment's behavior.

- *Frequency* is how often a bump arrives per unit time. A 10 Hz bump pattern arrives ten times a second (chatter). A 0.5 Hz bump pattern arrives once every two seconds (potholes). "Closed-loop sensitivity at frequency $\omega$" is how much a disturbance arriving at rate $\omega$ shakes the controlled output. Sensitivity is small where the loop rejects disturbances well, and large where it amplifies them.

- *Sensitivity at a given frequency* $\omega$ is therefore a number, between roughly 0 and some maximum. Plot it against $\omega$ and you get the *sensitivity curve*—a picture of how the

closed loop responds to disturbances at each frequency.

- *The conservation law* says: *you cannot make the sensitivity curve small everywhere.* Push it down at one band of frequencies, and it has to bulge up at another. The kid trading chatter for potholes is a low-bandwidth biological version of this; the Bode integral is the high-precision frequency-domain version.
- *Unstable poles* are the kicker. If the plant being controlled is *open-loop unstable*—meaning, if left alone with no controller, the plant would blow up rather than settle—then the conservation law gets stricter. The bulging has to be at least $\pi$ times the sum of the unstable poles. There is more total shaking when the plant wants to run away on its own.

**Where it came from.** The conservation law was first written down by Hendrik Wade Bode, an electrical engineer at Bell Labs, in his 1945 textbook *Network Analysis and Feedback Amplifier Design*—one of the foundational documents of twentieth-century engineering. It has been taught in every advanced undergraduate and graduate control-systems course in the world since at least the 1960s. Engineers learn it, design against it, and silently swear at it. The integral is the wall every controller designer eventually hits: *you cannot have it all at every frequency.*

**The geometric reading.** What this book argues—and what the rest of the chapter walks through—is that the Bode integral is not really a statement about frequency response. It is a statement about a *topological obstruction* in the closed-loop system, expressed in the only mathematical language available to Bode in 1945, which was the language of contour integrals on the complex plane. The integral is what an obstruction theorem looks like when you write it down using complex analysis instead

of differential geometry. Reframed in the differential-geometric language Chapter 3 set up, the Bode integral starts to look like a familiar object: a Gauss–Bonnet-style statement, with $\log |S(j\omega)|$ playing the role of a curvature density on the frequency line and the unstable poles playing the role of topological features that the integral has to account for. The framework reads it as one instance of the same kind of obstruction theorem that shows up in the Davis Duality and the Davis Field Equation.

Bode was doing differential geometry without knowing it. We can now read his result with the vocabulary he did not have.

**What the chapter does, in order.** Section 7.1 defines the sensitivity function $S(j\omega)$ and its complement, with each piece introduced in plain English before the math. Section 7.2 writes down the Bode integral itself and explains why the integral must be conserved. Section 7.3 walks the proof (Cauchy's residue theorem applied to $\log L(s)$ on a semicircular contour). Section 7.4 reframes the integral as a Gauss–Bonnet-style obstruction. Section 7.5 introduces the *Curvature Floor* inequality—the Davis-Geometric specialization of the integral for LTI systems—and Section 7.6 walks the ICARUS sidebar showing the integral as a runtime database query against the state-history bundle. The adaptive extension (where the inequality becomes strict and the right-hand side acquires curvature terms in the parameter manifold) is in Chapter 13.

§ 7.1

## *The sensitivity function and its complement*

Fix a single-input, single-output (SISO) closed-loop system with plant transfer function $P(s)$, controller $K(s)$, and unity-gain negative feedback. The loop transfer function is

$$L(s) \;=\; P(s)\,K(s),$$

and the closed-loop transfer from reference $r$ to output $y$ is

$$T(s) \;=\; \frac{L(s)}{1+L(s)},$$

called the *complementary sensitivity function*. The closed-loop transfer from output-disturbance $d$ to output $y$ is

$$S(s) \;=\; \frac{1}{1+L(s)},$$

called the *sensitivity function*. By construction,

$$S(s)+T(s) \;\equiv\; 1.$$

The identity $S+T=1$ is the chapter's first conservation law. It says: at every frequency, the closed-loop transfer to disturbances plus the closed-loop transfer to references sums to a constant. You cannot make both small at the same frequency. You can trade them off, but you cannot win on both simultaneously.

For control design, this means: at low frequencies you usually want $|T| \approx 1$ (the closed-loop follows the reference) and $|S| \approx 0$ (disturbances are rejected). At high frequencies you usually want $|T| \approx 0$ (high-frequency noise on the reference is rejected) and $|S| \approx 1$ (the closed-loop just lets through whatever the plant does

at high frequency, because trying to suppress it would require absurd gains). The identity $S + T = 1$ guarantees that you can transition between these regimes but not collapse them both to zero.

The sensitivity function $S$ is the design object of most interest because it also captures the closed-loop response to plant uncertainty. If the actual plant differs from the design plant by a multiplicative perturbation, the closed-loop perturbation scales with $T \cdot \Delta P/P$; robustness margins are set by bounds on $T$. If the disturbance enters at the output, the closed-loop response scales with $S \cdot d$; disturbance rejection is set by bounds on $S$. The two functions, $S$ and $T$, are the two quantities every classical control design specification is expressed in terms of.

§ 7.2

## *The Bode plot as gauge-invariant presentation*

The Bode plot of a transfer function $G(s)$ is the pair of graphs showing $20 \log_{10} |G(j\omega)|$ in decibels and $\arg G(j\omega)$ in degrees, plotted against $\omega$ on a logarithmic frequency axis. The plot is, mechanically, just an evaluation of $G$ along the positive imaginary axis $s = j\omega$, with magnitude expressed in $\log_{10}$ scale.

For the open-loop $L(s)$, the closed-loop $T(s)$, and the sensitivity $S(s)$, the corresponding Bode plots together completely characterize the closed-loop frequency response.

**The Bode plot is gauge-invariant.** Recall from §6, §6.6, that a transfer function $G(s)$ is one of the gauge-invariant quantities of a state-space model: under any similarity transformation of the state coordinates, the matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ transform but $G(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}$ is preserved. The Bode plot is

therefore a gauge-invariant graphical presentation of the closed-loop system.

This is the structural reason Bode plots are useful for design. A specification expressed in Bode form—"gain margin at least 6 dB," "phase margin at least 45°," "crossover frequency near 5 rad/s"—is gauge-invariant. Two engineers who specify in this language, regardless of their state-space coordinate conventions, are specifying the same closed-loop behavior. Gauge-dependent specifications ("set the (2,4) entry of $\mathbf{K}$ to 0.3") require the recipient to know the gauge in which the value is meaningful, which they usually do not.

**What the Bode plot reads as a section.** In the bundle language of Chapter 5, the Bode plot is a presentation of one section of the controller bundle—the LTI closed-loop section—restricted to the imaginary axis of the complex plane. The transfer function $G(s)$ is the section over the full complex plane (extended to a meromorphic function on the Riemann sphere). The Bode plot is the boundary trace of that section along the imaginary axis. The Bode plot is, in this sense, a one-dimensional slice through a two-dimensional bundle section, and the slice is chosen for the operational reason that the imaginary axis is where the steady-state frequency response lives.

The integral identity I am about to state is the consequence of completing the boundary trace into a closed contour. Cauchy's residue theorem then connects the contour integral to the meromorphic structure of $G(s)$—specifically, to the poles inside the contour. This is where the unstable open-loop poles enter the story.

§ 7.3

## *The Bode sensitivity integral*

**Theorem (Bode, 1945).** *Let the open-loop transfer function $L(s)$ be a proper rational function with relative degree $\geq 2$ (more poles than zeros, by at least two). Assume the closed loop is internally stable and has no imaginary-axis pole-zero cancellations. Let $p_1^+, \ldots, p_m^+$ be the right-half-plane poles of $L(s)$, counted with multiplicity. Then the closed-loop sensitivity function $S(s) = 1/(1 + L(s))$ satisfies*

$$\int_0^\infty \log|S(j\omega)|\, d\omega \;=\; \pi \sum_{i=1}^{m} \mathrm{Re}(p_i^+).$$

The integral is over positive frequencies (the integrand is even in $\omega$, so the integral over the whole real line is twice this). The proof is in any robust-control textbook (Doyle, Francis, Tannenbaum 1992; Skogestad and Postlethwaite 2005). The essential trick is to consider a large semicircular contour in the right half-plane, evaluate $\oint \log S(s)\, ds$ by residues, and let the semicircular radius go to infinity. The contributions from the imaginary axis and the semicircle separate; the relative-degree-$\geq 2$ hypothesis is exactly what makes the semicircular contribution vanish in the limit, leaving the imaginary-axis integral and the residue sum on equal footing. The residue contribution then counts exactly the right-half-plane poles of $L$.

**What the theorem says in plain English.** The integral of the log of the sensitivity magnitude, over all positive frequencies, is fixed. It depends on the plant's right-half-plane open-loop poles (and on nothing else, in the LTI case). The controller can shape the local distribution of $\log|S(j\omega)|$ across frequencies, but the total integral is a conservation law.

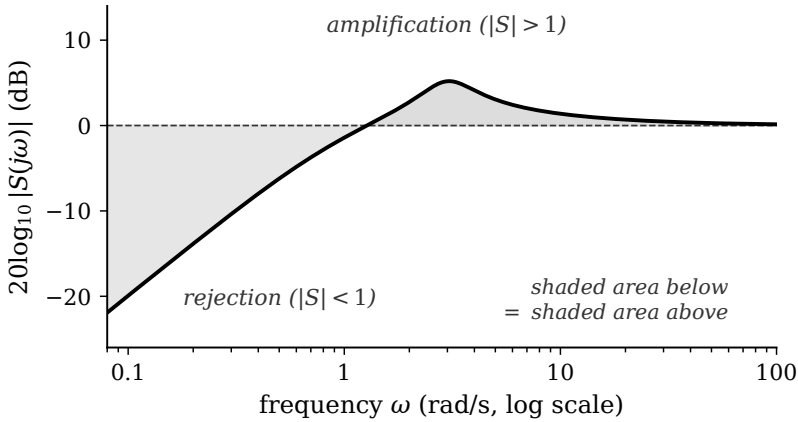



Figure 7.1: The Bode sensitivity function $|S(j\omega)|$ on log-frequency axes. Below the 0 dB line: the rejection band, where the loop attenuates disturbances ($|S| < 1$). Above the 0 dB line: the amplification band, where the loop amplifies them ($|S| > 1$). For a plant with no right-half-plane poles, the shaded area below equals the shaded area above—the conservation law in pictorial form. Pushing $|S|$ down at one band of frequencies forces a bulge somewhere else; the total integrated $\log |S|$ is fixed.

**The waterbed effect.** The Bode integral has been informally called the "waterbed effect." If the open-loop plant is stable (no right-half-plane poles), the integral is zero. Since $\log|S(j\omega)|$ is positive where $|S| > 1$ and negative where $|S| < 1$, the integral being zero means: the area above the $|S| = 1$ line equals the area below it. Push $|S|$ below 1 at low frequencies (good disturbance rejection in the operating band), and you must have $|S| > 1$ somewhere else (amplification at high frequencies or near the crossover). The areas balance. The peaks of $|S|$ are unavoidable.

If the open-loop plant is unstable, the integral is strictly positive: the area above the $|S| = 1$ line strictly exceeds the area below. The peaks of $|S|$ are not just unavoidable; they are larger by a fixed amount determined by the unstable pole locations. This is the price of stabilizing an unstable plant: bigger peaks in the sensitivity function, located somewhere in the frequency band, and there is no controller that can avoid them.

**The kid's wall was a conservation law of the same kind.** The lawnmower-bike of §1.0 had a fixed amount of capacity to spend: thirty feet of driveway. The kid could redistribute that capacity (start slow, build speed, or start fast and brake hard) but she could not increase it. The wall did not move. An unstable plant has the same structure: the integrated $\log|S|$ is fixed by the right-half-plane poles, and the controller can redistribute the peaks across frequency but cannot reduce the total. The kid intuited this without writing the integral. She looked at the driveway, looked at the bike, and concluded *no controller can avoid the wall if I start the engine.* The Bode integral is the formal statement of that same kind of conservation, with the wall replaced by the right-half-plane open-loop poles and the driveway replaced by the frequency axis.

**What this means for flight control.** Many aerospace plants are open-loop unstable. The F-16 longitudinal model is short-

period stable but phugoid lightly damped; the F-16 lateral subsystem has dutch-roll modes that hover near the imaginary axis and need active damping. Some aircraft—the X-29, the F-117, several uncrewed-aircraft platforms—are statically unstable longitudinally and require continuous active stabilization. For all of these, the Bode integral guarantees that any stabilizing controller produces a sensitivity function with a peak somewhere, and the peak's lower bound is set by the unstable open-loop pole locations.

This is why aerospace robustness margins (gain margin, phase margin, sensitivity peak $M_S$) are non-trivial design constraints, not just textbook bookkeeping. They are paying for the right-half-plane structure of the plant.

§ 7.4

## *The Bode integral shares the structural pattern of Gauss–Bonnet*

Now the geometric reading. I have been promising it since §7.0. Here it is.

Recall the Gauss–Bonnet theorem from differential geometry: for a compact two-dimensional Riemannian manifold $\Sigma$ without boundary,

$$\int_{\Sigma} K \, dA \;=\; 2\pi \, \chi(\Sigma),$$

where $K$ is the Gaussian curvature, $dA$ is the surface area element, and $\chi(\Sigma) = 2 - 2g$ is the Euler characteristic ($g$ is the genus, i.e. the number of handles). The integral of curvature over the whole surface is fixed by the topology. Adding a handle changes the integral by $-4\pi$; removing one changes it by $+4\pi$. The local

curvature distribution can be redistributed by a diffeomorphism, but the total is a topological invariant.

Now look again at the Bode integral:

$$\int_0^\infty \log|S(j\omega)|\,d\omega \;=\; \pi \sum_{i=1}^{m} \mathrm{Re}(p_i^+).$$

The integrand $\log|S(j\omega)|$ plays the role of a curvature 1-form on the (one-dimensional) frequency line. The right-hand side is $\pi$ times a count of right-half-plane poles—an integer-valued topological invariant of the closed-loop system, with the unstable pole locations weighted by their real parts. Just as the Euler characteristic counts handles, the unstable-pole sum counts a closed-loop topological feature: the obstruction to the system being internally stable from a single one-degree-of-freedom controller.

**The analogy is precise enough to write down.**

- Gauss–Bonnet base: compact 2-manifold $\Sigma$. Bode base: imaginary axis of $\mathbb{C}$ (with the appropriate compactification).
- Gauss–Bonnet curvature: Gaussian curvature $K(p)$, a scalar field on $\Sigma$. Bode curvature: $\log|S(j\omega)|$, a scalar field on the frequency line.
- Gauss–Bonnet topological invariant: Euler characteristic $\chi(\Sigma) = 2 - 2g$ (integer-valued, deformation invariant). Bode structural invariant: $\sum_i \mathrm{Re}(p_i^+)$, a real-valued weighted sum over right-half-plane poles. The Gauss–Bonnet quantity is a literal topological invariant; the Bode quantity is its structural analogue—fixed by the open-loop's right-half-plane spectrum but continuous in the pole locations, so an analogy of topological character rather than a literal topological invariant.

- Gauss–Bonnet constant: $2\pi$. Bode constant: $\pi$.
- Gauss–Bonnet conservation: redistributing $K$ across $\Sigma$ does not change $\int K\, dA$. Bode conservation: redistributing $\log|S|$ across frequency does not change $\int \log|S|\, d\omega$.

The analogy is not literal in the sense that the imaginary axis is not a compact 2-manifold. (It is a 1-manifold, and not even compact; the integral requires a relative-degree assumption to be finite.) And the right-hand side is real-valued, continuous in the pole locations, where the Gauss–Bonnet right-hand side is integer-valued and deformation-invariant. So the analogy is structural, not literal: *a local geometric quantity integrates over a base to give a structural invariant of the system, fixed by the system's spectral data (in Bode's case) or its topology (in Gauss–Bonnet's case).* This is what an obstruction theorem looks like in either language.

**The reading the chapter is committed to.** The Bode integral can be read, in the language of this book, as the frequency-domain expression of a curvature-like obstruction. The right-half-plane poles of the open-loop plant play the role of the obstructions; the sensitivity function $S(j\omega)$ plays the role of the local curvature distribution that the controller can redistribute but not eliminate; the integral identity is the conservation law that links local distribution to a global integer invariant. The waterbed effect, in this reading, is the analog of "you cannot smooth out the curvature of a torus into a sphere" for closed-loop control systems—an analogy, not a literal mathematical identification.

This reading is not a metaphor: the two statements share the same structural pattern, a local geometric quantity integrating over a base to give an integer-valued topological invariant of the system. Let me be precise about what I am and am not claiming. I am *not* claiming the Bode integral is literally a Gauss–Bonnet statement in the strict differential-geometric sense; the

complex-analytic proof goes through Cauchy's residue theorem and uses the meromorphic structure of $L(s)$ in a way that has no direct analog in the standard proof of Gauss–Bonnet. The two are sibling results, not the same result in two different languages. What I *am* claiming is that recognizing the parallel—local geometric quantity integrating to a topological invariant—is what lets the framework read both of them as obstruction theorems of the same general kind. The pedagogical value is in the recognition; the formal mathematical identification would be a separate paper.

The Davis-Geometric framework reads the Bode integral as a sibling of its own obstruction theorems. The next section makes the inequality form explicit.

§ 7.5

## *The Curvature Floor inequality, LTI case*

The Davis-Geometric framework's general inequality is

$$\int_0^\infty \log|S(j\omega)|\,d\omega \;\geq\; \pi\sum_{i=1}^{m}\mathrm{Re}(p_i^+) \;+\; \alpha\,\mathrm{tr}(\mathbf{\Gamma}^{-1}) \;+\; \beta\,\mathrm{curv}(\bar{\Theta}),$$

called the *Curvature Floor inequality.* The right-hand side has three pieces:

- The classical Bode term $\pi\sum\mathrm{Re}(p_i^+)$. Spectral cost of unstable open-loop poles—structurally analogous to a topological obstruction term, but continuous in the pole locations rather than integer-valued.
- An adaptation-gain term $\alpha\,\mathrm{tr}(\mathbf{\Gamma}^{-1})$, where $\mathbf{\Gamma}$ is the adaptive controller's gain matrix. Cost of running with adaptive

feedback at all.

- A parameter-manifold curvature term $\beta\,\mathrm{curv}(\bar{\Theta})$, where $\mathrm{curv}(\bar{\Theta})$ is a scalar curvature obstruction on the parameter manifold $\bar{\Theta}$ over which adaptation runs. Cost of having a curved parameter space.

For an LTI system without adaptation, the second and third terms are zero, and the inequality reduces to the Bode equality:

$$\boxed{\int_0^\infty \log|S(j\omega)|\,d\omega \;=\; \pi\sum_{i=1}^{m}\mathrm{Re}(p_i^+).}$$

This is the LTI Curvature Floor. The integral is bounded below by exactly the Bode integral, and the bound is achieved (the inequality is an equality) for any LTI controller.

**What changes when adaptation enters.** Chapter 13 develops the full inequality. Briefly, for orientation: when the controller is adaptive (gains evolve in time based on tracking error, parameter estimation, or both), the adaptive law introduces additional disturbance-amplification structure that shows up as positive contributions to $\int \log|S(j\omega)|\,d\omega$ that are not present in the LTI case. The adaptation gain trace $\mathrm{tr}(\mathbf{\Gamma}^{-1})$ controls one piece of this; the curvature of the parameter manifold $\bar{\Theta}$ controls another. Both are positive, so the inequality is strict for adaptive systems.

The framework's claim is that the right-hand side is the right way to budget for adaptation. Aerospace engineers know empirically that "adaptive controllers are harder to certify than LTI ones" and "high adaptive gains amplify high-frequency disturbances." The Curvature Floor inequality is the framework's quantitative form of these heuristics: the cost of adaptation is at least $\alpha\,\mathrm{tr}(\mathbf{\Gamma}^{-1}) + \beta\,\mathrm{curv}(\bar{\Theta})$ in the integrated $\log|S|$ budget, and the constants $\alpha, \beta$ depend on the specific adaptive law.

The Chapter 13 sidebar notes that quantitative fitting of $\alpha$ and $\beta$ was not attempted in the empirical work to date; the conjecture remains analytically open, and the empirical validation showed only qualitative consistency with the predicted monotonicity. The inequality form is a structural claim of the framework; the constants are a future-work item.

For this chapter—the LTI chapter—only the Bode equality matters. The full inequality is the forward pointer.

§ 7.6

## *Sidebar: ICARUS implementation of the closed-loop sensitivity*

*The following sidebar is reproduced from the ICARUS architecture documentation. It is a small illustration of how the framework's substrate makes the chapter's content operational, computable, and falsifiable.*

> ICARUS computes the closed-loop sensitivity function as a `SPECTRAL` query against the state-history bundle. The Curvature Floor inequality is declared as an `INVARIANT` at the schema layer. Violations during ingest are rejected by GIGI's invariant enforcement; the constraint is encoded into the substrate, not added as a runtime check.

**What this means concretely.** In a traditional flight-control software stack, the sensitivity function $|S(j\omega)|$ is something an analyst computes offline against a linearized plant model, plots, eyeballs for peaks, and then archives. The Bode integral is a

property of the offline-computed sensitivity, checked once during certification, and never revisited at flight time.

In ICARUS, $|S(j\omega)|$ is a query. The state-history bundle holds the closed-loop response data from recent flight; the `SPECTRAL` query computes the empirical sensitivity function over a configurable frequency window. The Curvature Floor invariant—$\int_0^\infty \log|S(j\omega)|\,d\omega \geq \pi \sum \mathrm{Re}(p_i^+)$—is checked at every ingest. If a state-history sample would violate the invariant, GIGI's invariant enforcement rejects the sample (and the system flags an anomaly: most likely, the plant's open-loop pole estimate is wrong, or the controller has lost stability margin). The constraint is encoded into the substrate.

The advantage of doing this in the substrate, rather than as a runtime check on top, is that violations cannot get past ingest. The flight-control loop cannot enter a closed-loop state for which $\int \log|S|\,d\omega$ is below the Bode floor, because such a state would have been rejected at ingest before the controller could act on it.

This is a small claim. The Bode integral is not the only constraint a flight-control system has to respect. But it is the right kind of constraint to have at the schema layer rather than at the runtime layer: the constraint is a structural invariant of the closed-loop system (continuous in pole locations but fixed by the right-half-plane spectrum), and such invariants belong in the substrate where they cannot be violated at runtime.

§ 7.7

## *What you take from this chapter*

Three things.

**First.** The sensitivity function $S(s) = 1/(1 + L(s))$ and its complement $T(s) = L(s)/(1 + L(s))$ are the two quantities every classical control specification is built from. They satisfy $S + T = 1$ identically. Robustness margins, disturbance rejection, and reference tracking are all expressed in terms of bounds on $|S|$ and $|T|$.

**Second.** The Bode sensitivity integral

$$\int_0^\infty \log|S(j\omega)|\, d\omega \;=\; \pi \sum_i \mathrm{Re}(p_i^+)$$

is a conservation law. It says: integrated $\log|S|$ over all positive frequencies is fixed by the right-half-plane open-loop poles. Push $|S|$ down here, it bulges up there. For unstable plants, the integral is strictly positive: the bulging cannot be eliminated, only redistributed.

**Third.** The Bode integral is a Gauss–Bonnet–style statement (a structural analogy, not a literal topological identity: the imaginary axis is not a compact 2-manifold, and the right-hand side is real-valued rather than integer-valued). The integrand $\log|S(j\omega)|$ is a curvature 1-form on the frequency line; the right-hand side is $\pi$ times the structural invariant $\sum_i \mathrm{Re}(p_i^+)$ over right-half-plane poles. The Bode result is what an obstruction theorem looks like in the complex-analysis language available in 1945. In the geometric language of this book, it is the LTI specialization of the Curvature Floor inequality

$$\int \log|S|\, d\omega \;\geq\; \pi \sum \mathrm{Re}(p_i^+) \;+\; \alpha\, \mathrm{tr}(\mathbf{\Gamma}^{-1}) \;+\; \beta\, \mathrm{curv}(\bar{\Theta}),$$

whose full form is developed in Chapter 13.

ICARUS computes the sensitivity function as a database query, declares the Curvature Floor as a schema invariant, and rejects violating samples at ingest. The constraint Bode wrote down by hand in 1945 lives in the substrate.

Chapter 8 takes the same plant and applies optimal control: the LQR problem. The Curvature Floor reappears there in a different guise: LQR's algebraic Riccati equation is computing, implicitly, a balance between the same topological obstruction and a quadratic cost, and the resulting closed-loop sensitivity satisfies the same Bode constraint without anyone having written it down. The geometric reading lets us name what the LQR algebra is doing.

*Vamos.*

Chapter 8

# *LQR, Riccati, and curvature-gated modulation*

*An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*
*Richard Bellman, Dynamic Programming (1957), the Principle of Optimality*

§ 8.0

## *The optimal section*

Go back to the driveway. The kid is sitting on the bike with the engine bolted on, hand near the starter, running the trajectory forward in her head. She is doing something at that moment that has a name in control theory; she does not know the name yet. She is weighing two costs against each other—the cost of *not* riding (boredom, wasted afternoon, parts she will have to put back) against the cost of *riding into the wall* (injury, her

mother's face). The weighing is not random. She is integrating: thinking about what happens over the next ten seconds if she pulls the starter, the next thirty seconds, the next minute. She is summing imagined cost over an imagined future and asking which choice gives the smaller total. The control input that wins (don't pull the starter) is the one with the lower integrated cost.

That is the operation this chapter is about. The control-theory name for it is the **Linear Quadratic Regulator**—LQR for short. The book will unpack the name one word at a time, but here is the picture first: LQR is the cleanest possible version of the kid's calculation. You give it a model of how the system moves, you give it a way of scoring how much you dislike each kind of deviation, and it gives you back, in closed form, the optimal feedback law—the rule that says "given where you are right now, here is what to do next."

**Unpacking the name, one word at a time.**

- *Linear* means the plant—the thing being controlled—moves according to a linear rule. The current rate-of-change of the state is a linear combination of the current state and the current control input. The bike's small-deviation dynamics are linear near a chosen reference; the F-16's longitudinal dynamics are linear near a chosen trim point; most engineering systems are linear in a neighborhood of their operating point.
- *Quadratic* means the cost function—the thing being minimized—is built from squared deviations. Twice the deviation from the desired state costs four times as much, not twice. This is what makes the answer come out clean: quadratic cost has a closed-form minimum.
- *Regulator* means the goal is to drive the state to zero (or, equivalently, to a constant reference) and hold it there.

"Regulate" as in "regulate the temperature of the room." The kid's regulator goal was to keep her body's velocity and position bounded; the F-16's regulator goal is to hold a commanded altitude and airspeed.

- *Time-invariant* means the rule doesn't change with time. Same physics at 9 AM as at 5 PM. Real systems have slow time variation (the bike's tire pressure drops, the F-16's mass decreases as it burns fuel); the LQR treatment assumes those changes are slow enough to ignore on the controller's timescale, and you re-solve the equation when they aren't.

- *Infinite horizon* means the cost integral runs from now to forever, not from now to some fixed final time. The kid was implicitly integrating to forever—she did not care whether the wall was 5 seconds away or 50, only that the total cost of hitting it was very large. Infinite-horizon optimal control gives a fixed feedback rule (the rule does not depend on time); finite-horizon optimal control gives a time-varying rule. LQR is the fixed-rule case.

**Where the name's pieces come from historically.** The closed-form result for the infinite-horizon LQR problem was given by Rudolf Kalman in 1960[1]—the same year as his more famous filtering paper. The optimal feedback turns out to depend on a single matrix that solves a single algebraic equation; the equation is named after Jacopo Riccati, the eighteenth-century Italian mathematician whose first-order nonlinear ODE[2] has the same

[1] Kalman, R.E. "Contributions to the Theory of Optimal Control," *Boletín de la Sociedad Matemática Mexicana* 5(2), 102–119 (1960). This is the LQR paper. Kalman's better-known filtering paper, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering* 82(1), 35–45 (1960), appeared the same year in a different venue. Both contributions emerged from the same year's work and the same conceptual program: state-space methods as the right framework for optimal estimation and control.

[2] The classical Riccati equation, published in 1724, has the form $dy/dx =$

quadratic structure that turns up here, generalized from scalars to matrices and from differential to algebraic form. The practical solution method most engineers actually use was published by David Kleinman in 1968; a version of it was already running on the Apollo guidance computer by then, doing optimal control of the spacecraft's attitude on the same code that ran Chapter 14's strapdown integration.

**What the chapter does, in order.** Section 8.1 writes the LQR problem down as a mathematical statement, introducing each symbol when it first appears. Section 8.2 writes the Riccati equation that determines the optimal feedback. Section 8.3 shows how the equation is solved in practice (Newton–Kleinman iteration). Section 8.4 introduces the geometric reading the rest of this book uses—the optimal value function as a Riemannian metric on the state manifold. Sections 8.5 and 8.6 introduce the framework's curvature-gated modulation $m(\hat{\kappa}) \cdot \mathbf{K}_{\text{lqr}}$ and walk what the modulation buys and what it costs.

This chapter is the formal version of what the kid did. The plant comes from Newton's laws and the geometry of her body and the bike. The cost weights come from her preferences over outcomes. The Riccati equation falls out of the constraint that the integrated cost be minimized over admissible inputs. The whole calculation, in her case, took two seconds. The whole calculation, for an F-16 longitudinal controller, takes Newton–Kleinman fifteen iterations on a laptop. Same structure, different scale.

---

$q_0(x) + q_1(x)\, y + q_2(x)\, y^2$. The quadratic term in $y$ is what gives the equation its characteristic shape and its name; the LQR's quadratic term $\mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}$ is the matrix analog. The connection to control-theoretic LQR is structural rather than historical—Riccati had no idea anyone would ever feed back a state—but the algebraic skeleton is the same.

§ 8.1

# *The LQR problem, written down*

This section writes LQR as a mathematical statement, introducing each symbol with a plain-English meaning before any equation appears. The reader who finds equations easier than prose is welcome to skim the prose and read the equations; the reader who finds the prose easier is welcome to skim the equations and read the prose. The two sides say the same thing.

**The state.** The *state* of the system is the list of numbers that describes where the system is right now. For the kid on the bike: her forward speed, her lean angle, her steering angle, her position on the road. For the F-16 longitudinal model from Chapter 6: forward speed, angle of attack, pitch rate, pitch angle. The state is written $\mathbf{x}$ and lives in $\mathbb{R}^n$ where $n$ is the number of state variables. For a bike, $n$ might be 4 or 6; for an F-16, $n = 4$ (the longitudinal channel).

**The control input.** The *control input* is the list of numbers describing what the controller is doing right now. For the bike: handlebar torque, brake pressure. For the F-16: elevator deflection (one number). The control is written $\mathbf{u}$ and lives in $\mathbb{R}^m$ where $m$ is the number of control variables. For an F-16 longitudinal controller, $m = 1$.

**The plant equation.** The *plant equation* says how the state changes when a control is applied. In linear form:

$$\dot{\mathbf{x}}(t) \;=\; \mathbf{A}\,\mathbf{x}(t) \;+\; \mathbf{B}\,\mathbf{u}(t), \qquad \mathbf{x}(0) = \mathbf{x}_0.$$

The dot means time-derivative. The matrix $\mathbf{A}$ is $n \times n$ and says how the state's current value contributes to its rate of change (the natural drift of the plant when no control is applied). The matrix

$\mathbf{B}$ is $n \times m$ and says how the current control input contributes to the state's rate of change (where the control enters). The starting state is $\mathbf{x}_0$.

For the bike: $\mathbf{A}$ encodes how forward speed couples to lean angle through gyroscopic effects, how lean angle decays through gravity, etc. $\mathbf{B}$ encodes how handlebar torque enters the steering channel. The matrices come from Newton's laws plus the geometry of the bike's mass distribution.

**The cost functional.** The *cost functional* scores how much you dislike the trajectory. For LQR it has the specific form

$$J[\mathbf{x}(\cdot), \mathbf{u}(\cdot)] \;=\; \int_0^\infty (\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} \;+\; \mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u})\, dt.$$

Two pieces, integrated over all future time.

The first piece, $\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x}$, is the *state cost*: a quadratic score on how far the state is from zero. The matrix $\mathbf{Q}$ is $n \times n$, symmetric, with non-negative eigenvalues (positive semi-definite, written $\mathbf{Q} \succeq 0$). The diagonal entries of $\mathbf{Q}$ say how much you dislike each individual state being nonzero; the off-diagonal entries say how much you dislike specific combinations. For the bike: a large $Q$-entry on lean-angle squared says the kid really dislikes leaning over; a small $Q$-entry on forward-speed deviation says she does not mind small speed errors. The cost rewards keeping each state component close to zero, weighted by $\mathbf{Q}$.

The second piece, $\mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u}$, is the *control cost*: a quadratic score on how hard the controller is working. The matrix $\mathbf{R}$ is $m \times m$, symmetric, with strictly positive eigenvalues (positive definite, written $\mathbf{R} \succ 0$). Big $R$ means control is expensive (a hard yank on the handlebars hurts); small $R$ means control is cheap (you can move the elevator freely). The cost rewards using small control inputs, weighted by $\mathbf{R}$.

The $\int_0^\infty$ means: integrate this cost from now to forever. A

trajectory that takes longer to settle costs more, because the integrand stays nonzero longer.

**The problem.** Given the plant matrices $(\mathbf{A}, \mathbf{B})$ and the cost weights $(\mathbf{Q}, \mathbf{R})$, find the feedback rule $\mathbf{u}(t) = \mathbf{u}^*(\mathbf{x}(t))$—a function of the current state—that makes $J$ smallest. That is the LQR problem.

**Three conditions that have to hold.** The problem has a clean answer when three structural conditions are met. Each has a plain-English meaning before its formal name.

- *Stabilizability of* $(\mathbf{A}, \mathbf{B})$: *some feedback exists that can make the closed-loop stable.* Equivalently, the unstable modes of $\mathbf{A}$ (the directions the plant blows up in on its own) are reachable through $\mathbf{B}$ (the controller can push on them). If a plant has an unstable mode that the controller cannot touch, the cost integral diverges and LQR cannot save you.
- *Detectability of* $(\mathbf{A}, \mathbf{Q}^{1/2})$: *every unstable mode shows up in the cost.* If a plant has an unstable mode that does not appear anywhere in the state-cost matrix $\mathbf{Q}$, the optimizer cannot see that mode growing and the optimal feedback would let it grow without penalty. Detectability rules this out.
- *Strictly positive control cost,* $\mathbf{R} \succ 0$: *control is genuinely costly in every direction.* If $\mathbf{R}$ were degenerate, the optimizer could send unbounded control along a free direction, and the answer would be pathological.

Under these three conditions, the problem has a clean answer:

$$\mathbf{u}^*(\mathbf{x}) \;=\; -\mathbf{K}_{\text{lqr}}\,\mathbf{x}, \qquad \mathbf{K}_{\text{lqr}} \;=\; \mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}.$$

The optimal control is a fixed gain $\mathbf{K}_{\text{lqr}}$ (size $m \times n$) multiplied by the negative of the current state. The gain $\mathbf{K}_{\text{lqr}}$ is built from the

cost-weight matrix $\mathbf{R}$, the plant's control-entry matrix $\mathbf{B}$, and one new matrix $\mathbf{P}$ that we have not seen yet. The new matrix $\mathbf{P}$ is what the next section is about.

**What this looks like at the controller.** The controller has the gain $\mathbf{K}_{\text{lqr}}$ stored in memory (computed once, offline, by solving for $\mathbf{P}$). At each control cycle, the controller reads the current state $\mathbf{x}(t)$, multiplies by $-\mathbf{K}_{\text{lqr}}$, and that is the control input to apply. No optimization at runtime—just one matrix multiply. This is why LQR is the workhorse of safety-critical aerospace control: predictable cost per cycle, no iteration, no chance of the solver failing to converge mid-flight.

The whole question of the chapter is now: how do you compute $\mathbf{P}$? That is what §8.2 takes up.

§ 8.2

## *The continuous-time algebraic Riccati equation*

The continuous-time algebraic Riccati equation (CARE) for the LQR problem is

$$\boxed{\mathbf{A}^{\mathrm{T}}\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P} + \mathbf{Q} \;=\; \mathbf{0}.}$$

This is a quadratic matrix equation in $\mathbf{P}$. It has multiple solutions in general; the LQR-relevant solution is the unique *stabilizing* one, the maximal solution under the partial order $\mathbf{P}_1 \succeq \mathbf{P}_2$. With $\mathbf{Q} \succeq 0$ and the standard stabilizability and detectability assumptions on the pair $(\mathbf{A}, \mathbf{B})$ and $(\mathbf{A}, \mathbf{Q}^{1/2})$, the stabilizing $\mathbf{P}$ is positive semidefinite; it is strictly positive definite when $(\mathbf{A}, \mathbf{Q}^{1/2})$ is observable, which is the case for the LQR designs in this book.

**Where the CARE comes from.** The optimal value function $V(\mathbf{x}) = \min_{\mathbf{u}(\cdot)} J[\mathbf{x}(\cdot), \mathbf{u}(\cdot) | \mathbf{x}(0) = \mathbf{x}]$ satisfies the Hamilton–Jacobi–Bellman equation

$$0 = \min_{\mathbf{u}} \Big[ \mathbf{x}^{\mathrm{T}} \mathbf{Q} \mathbf{x} + \mathbf{u}^{\mathrm{T}} \mathbf{R} \mathbf{u} + \nabla V(\mathbf{x}) \cdot (\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) \Big].$$

Trying $V(\mathbf{x}) = \mathbf{x}^{\mathrm{T}} \mathbf{P} \mathbf{x}$ for some symmetric positive-definite $\mathbf{P}$, the gradient is $\nabla V = 2\mathbf{P}\mathbf{x}$. Minimizing over $\mathbf{u}$ gives $\mathbf{u}^* = -\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{x}$, and substituting back into the HJB equation produces, after one symmetry step, exactly the CARE. The symmetry step is small but worth naming: the bilinear term $2\mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{A}\mathbf{x}$ that appears in the substitution equals $\mathbf{x}^{\mathrm{T}}(\mathbf{A}^{\mathrm{T}}\mathbf{P} + \mathbf{P}\mathbf{A})\mathbf{x}$ because $\mathbf{P}$ is symmetric and $\mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{A}\mathbf{x} = \mathbf{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ by transposing the scalar. This is what produces the CARE's symmetric form on the left-hand side.

The CARE is therefore the consistency equation for the assumed quadratic form of the value function. The matrix $\mathbf{P}$ is the Hessian of the optimal value function at the origin. The gain $\mathbf{K}_{\mathrm{lqr}} = \mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}$ is, up to a factor of 2, the gradient of $V$ projected onto the control direction.

**What the CARE says structurally.** The four terms have an interpretation:

- $\mathbf{A}^{\mathrm{T}}\mathbf{P} + \mathbf{P}\mathbf{A}$ is the Lyapunov-flow term: the rate of change of $\mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ along open-loop trajectories.
- $-\mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}$ is the control-effort reduction: how much $V$ decreases under optimal feedback.
- $\mathbf{Q}$ is the state-cost source: how much running cost accumulates per unit time per unit state-norm.
- The whole equation says: the rate of $V$-decrease under optimal control exactly balances the rate of running-cost accumulation, at every point of state space.

This is the algebraic content of the Principle of Optimality, specialized to quadratic cost and linear dynamics. The Riccati equation *is* the LQR problem.

**The closed-loop dynamics.** Substituting $\mathbf{u}^* = -\mathbf{K}_{\mathrm{lqr}}\mathbf{x}$ into the plant gives

$$\dot{\mathbf{x}} \;=\; (\mathbf{A} - \mathbf{B}\mathbf{K}_{\mathrm{lqr}})\mathbf{x} \;=\; (\mathbf{A} - \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P})\mathbf{x}.$$

Using the CARE, one can show that $\mathbf{A} - \mathbf{B}\mathbf{K}_{\mathrm{lqr}}$ is Hurwitz (all eigenvalues in the open left half-plane), so the closed-loop is asymptotically stable. This is the LQR closed-loop stability theorem; the proof is in any optimal-control text.

§ 8.3

## *Newton–Kleinman: how the CARE is solved in practice*

The CARE is a quadratic matrix equation. There is no closed-form solution in the entries of $\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}$; one has to compute $\mathbf{P}$ numerically. Several algorithms exist—the Schur method (Laub 1979) is the standard, with the eigenstructure of the Hamiltonian matrix $\left[\begin{smallmatrix} \mathbf{A} & -\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}} \\ -\mathbf{Q} & -\mathbf{A}^{\mathrm{T}} \end{smallmatrix}\right]$ providing $\mathbf{P}$ as the product of two of its invariant-subspace bases.

The algorithm that ICARUS uses, and that I want to walk through briefly, is older and more transparent: Newton–Kleinman iteration (Kleinman 1968).

**Newton–Kleinman.** Start with any stabilizing feedback $\mathbf{K}_0$ (one exists by the stabilizability hypothesis; in practice a pole-placement gain is the standard initializer). At iteration $i$:

1. Solve the Lyapunov equation

$$(\mathbf{A} - \mathbf{B}\mathbf{K}_i)^{\mathrm{T}}\mathbf{P}_i + \mathbf{P}_i(\mathbf{A} - \mathbf{B}\mathbf{K}_i) \;=\; -(\mathbf{Q} + \mathbf{K}_i^{\mathrm{T}}\mathbf{R}\mathbf{K}_i).$$

   This is a linear equation in $\mathbf{P}_i$ and is solved by standard Bartels–Stewart or Hammarling methods.

2. Update the gain:

$$\mathbf{K}_{i+1} \;=\; \mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}_i.$$

3. Repeat until $\|\mathbf{K}_{i+1} - \mathbf{K}_i\| < \epsilon$ for some tolerance.

Kleinman proved in 1968 that the iteration is exactly Newton's method applied to the CARE viewed as a quadratic operator equation. The convergence is therefore quadratic: the error roughly squares at each iteration, and in practice the iteration converges to machine precision in 5–15 steps.

**Why Newton–Kleinman and not Schur.** For the ICARUS use case, two properties of Newton–Kleinman matter. First, the iteration is incremental: if the plant model is updated slightly (different trim, different parameter estimate), the previous $\mathbf{P}$ is a good initial guess for the new CARE, and the iteration converges in one or two steps from a hot start. The Schur method does not benefit from a hot start in the same way. Second, the iteration is auditable: every step is a Lyapunov solve, and the closed-loop matrix $\mathbf{A} - \mathbf{B}\mathbf{K}_i$ is Hurwitz at every iteration. This means the iteration is producing stabilizing gains throughout, not converging through some non-stabilizing intermediate value. For a safety-critical application like flight control, intermediate-iterate stability is a structural property worth paying for.

§ 8.4

# *LQR gain is gauge-dependent; closed-loop eigenvalues are not*

This section makes §6.6's invariant/dependent distinction concrete on LQR. The gauge transformations are the ones named in §6.4: similarity changes of the state coordinates, scalings of the units, choice of body-axis frame.

**The transformation rule.** Under a similarity transformation $\mathbf{T}$ on the state ($\mathbf{x} \to \mathbf{T}\mathbf{x}$, so $\mathbf{A} \to \mathbf{T}\mathbf{A}\mathbf{T}^{-1}$, $\mathbf{B} \to \mathbf{T}\mathbf{B}$), the LQR cost matrices transform as $\mathbf{Q} \to \mathbf{T}^{-\mathrm{T}}\mathbf{Q}\mathbf{T}^{-1}$ (so that $\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x}$ is invariant in the new coordinates) and $\mathbf{R} \to \mathbf{R}$ (control coordinates unchanged). The CARE solution transforms as $\mathbf{P} \to \mathbf{T}^{-\mathrm{T}}\mathbf{P}\mathbf{T}^{-1}$, and the gain transforms as

$$\mathbf{K}_{\mathrm{lqr}} \;\longrightarrow\; \mathbf{K}_{\mathrm{lqr}}\,\mathbf{T}^{-1}.$$

This is the rule from §6.5. The closed-loop matrix $\mathbf{A} - \mathbf{B}\mathbf{K}_{\mathrm{lqr}}$ transforms as $\mathbf{T}(\mathbf{A} - \mathbf{B}\mathbf{K}_{\mathrm{lqr}})\mathbf{T}^{-1}$, preserving its eigenvalues.

**The operational consequence.** The numerical entries of $\mathbf{K}_{\mathrm{lqr}}$ depend on which state coordinates and units you computed in. They are not, by themselves, a meaningful specification of the controller. An engineer who hands you "$\mathbf{K}_{\mathrm{lqr}} = [0.45, -1.2, 0.08, 3.7]$" has told you nothing useful unless they also tell you the gauge in which that vector is meaningful.

What the engineer should hand you is the closed-loop spectrum, or the closed-loop transfer function, or the response to a canonical disturbance—any of the gauge-invariant quantities of §6.6. From those, you can compute $\mathbf{K}_{\mathrm{lqr}}$ in any gauge you choose.

**What the LQR cost weights specify.** The choice of $(\mathbf{Q}, \mathbf{R})$

is a design choice, not a physical fact. Different $(\mathbf{Q}, \mathbf{R})$ produce different optimal gains. The standard engineering heuristic, due to Bryson (1979), is to scale $\mathbf{Q}$ and $\mathbf{R}$ as inverse squares of the maximum allowable state and control excursions:

$$Q_{ii} \;\approx\; \frac{1}{x_{i,\max}^2}, \qquad R_{jj} \;\approx\; \frac{1}{u_{j,\max}^2}.$$

This is itself a gauge choice—a choice of how to weight the cost against the physical units of the states and controls. There is no gauge-invariant "correct" $(\mathbf{Q}, \mathbf{R})$; the choice is part of what the designer specifies, and different choices produce different (gauge-invariant) closed-loop spectra. The art of LQR design is, structurally, the art of picking a $(\mathbf{Q}, \mathbf{R})$ that produces a closed-loop spectrum you like, by iterating over candidate weights and recomputing the CARE solution.

§ 8.5

## *The geometric reading:* $\mathbf{P}$ *as a Riemannian metric*

The Davis-Geometric reading of LQR rests on a single observation: the matrix $\mathbf{P}$, which is the Hessian of the optimal value function $V(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ at the origin, defines a Riemannian metric on the state manifold near the trim.

The metric is

$$g_{\mathbf{P}}(\mathbf{x}, \mathbf{y}) \;=\; \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{y},$$

positive-definite because $\mathbf{P} \succ 0$. The optimal value function $V(\mathbf{x}) = g_{\mathbf{P}}(\mathbf{x}, \mathbf{x})$ is the squared length of $\mathbf{x}$ in this metric, up to a factor.

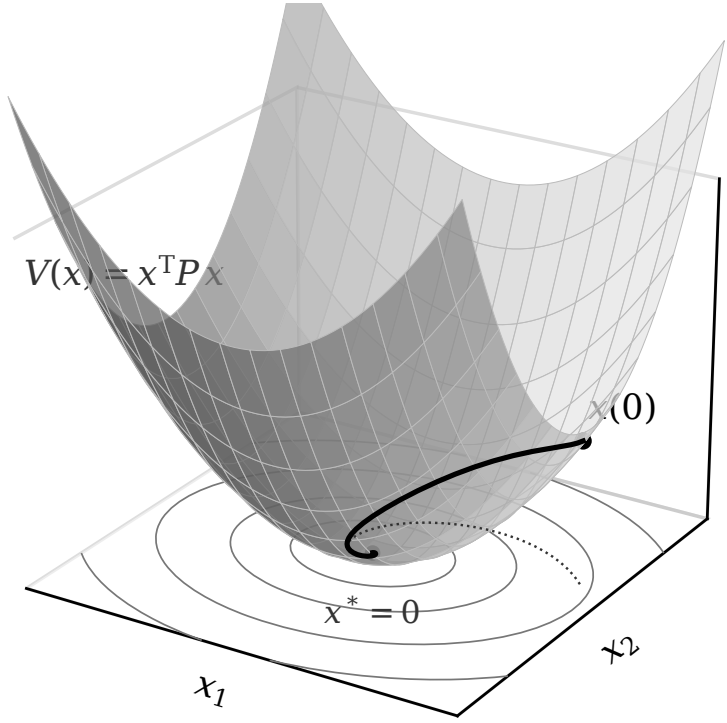



Figure 8.1: The LQR value function as a quadratic bowl. The optimal value function $V(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ is a paraboloid on the state space, with the equilibrium at its bottom. The closed-loop trajectory descends the bowl monotonically: at every instant the optimal control $\mathbf{u}^* = -\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ is the steepest descent on this landscape, in the $g_{\mathbf{P}}$ metric on states projected through the $\mathbf{R}^{-1}$ metric on controls. The dotted trace on the floor is the trajectory's shadow on the state plane.

The optimal control direction at state $\mathbf{x}$ is

$$\mathbf{u}^*(\mathbf{x}) \;=\; -\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{x},$$

which can be read as: take the gradient of $V$ in the $g_{\mathbf{P}}$ metric, project it onto the column space of $\mathbf{B}$ using the $\mathbf{R}^{-1}$ inner product on control space, and negate. The optimal control is, structurally, a (negative-)gradient descent on the value-function landscape, where the gradient is taken in the $g_{\mathbf{P}}$ metric on states and projected with the $\mathbf{R}^{-1}$ metric on controls.

**The CARE as a balance equation.** Read in this light, the CARE

$$\mathbf{A}^{\mathrm{T}}\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P} + \mathbf{Q} \;=\; \mathbf{0}$$

is a balance equation for the metric. The first two terms, $\mathbf{A}^{\mathrm{T}}\mathbf{P} + \mathbf{P}\mathbf{A}$, are the Lie derivative of the metric $g_{\mathbf{P}}$ along the open-loop flow $\mathbf{A}\mathbf{x}$—they measure how the open-loop dynamics distort the metric. The third term, $-\mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}$, is the gauge-projected control effort—how much of the available control authority reaches the metric. The fourth term, $\mathbf{Q}$, is the state-cost source—how much running cost the chosen weighting puts onto the metric.

The CARE says: the metric is in equilibrium when Lie-derivative distortion plus state-cost source equals gauge-projected control effort. The unique $\mathbf{P}$ that solves this is the optimal metric—the one for which the value function and the control law are mutually consistent under the Principle of Optimality.

**The closed-loop as gradient descent.** Under the optimal control, the closed-loop dynamics are

$$\dot{\mathbf{x}} \;=\; (\mathbf{A} - \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P})\mathbf{x}.$$

The rate of change of the value function along trajectories is

$$\frac{dV}{dt} \;=\; 2\mathbf{x}^{\mathrm{T}}\mathbf{P}\dot{\mathbf{x}} \;=\; -\mathbf{x}^{\mathrm{T}}(\mathbf{Q}+\mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P})\mathbf{x} \;<\; 0$$

(strict because $\mathbf{Q} \succeq 0$ and $\mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P} \succeq 0$, with the sum strictly positive on nonzero $\mathbf{x}$ by the detectability hypothesis). So the value function is strictly decreasing along trajectories, which is the Lyapunov-function statement of closed-loop stability.

This is what LQR is, geometrically: a closed-loop dynamics that performs gradient descent on a value-function landscape whose metric the CARE has tuned to balance state cost against control effort. The gradient descent is exact—the LQR controller does not just approximate gradient descent; the LQR controller *is* the gradient descent in the $g_{\mathbf{P}}$ metric.

**Why this matters for ICARUS.** In the framework's language: the LQR-optimal section $\mathbf{K}_{\mathrm{lqr}}$ of the controller bundle is the section that performs steepest descent on the value-function landscape in the metric $g_{\mathbf{P}}$. The framework's curvature-gated modulation, which the next section names, replaces this section with $m(C)\,\mathbf{K}_{\mathrm{lqr}}$—a scaled section that descends at a slower rate when the metric's local curvature obstruction is high. The modulation does not change which direction the controller pushes; it changes how aggressively. The descent is still in the right direction; it just respects the curvature when curvature warns it that the linearization is unreliable.

§ 8.6

## *Sidebar: ICARUS implementation of LQR*

*The following sidebar is reproduced from the ICARUS architecture documentation.*

> ICARUS does not solve CARE numerically at flight time. The optimal feedback gain $\mathbf{K}_{\text{lqr}}$ (computed offline via Newton–Kleinman against the linearized plant) is stored as a section of a controller bundle; gain modulation at flight time replaces $\mathbf{K}_{\text{lqr}}$ with $m(C) \cdot \mathbf{K}_{\text{lqr}}$ where $m(C) = m_{\min} + (m_{\max} - m_{\min}) \cdot C^{\gamma}$ is the curvature-gated gain-modulation schedule. CARE is solved once at design time.

**Unpacking the schedule.** The modulation function

$$m(C) \;=\; m_{\min} + (m_{\max} - m_{\min}) \cdot C^{\gamma}$$

maps the operational confidence $C \in [0, 1]$ to a scalar gain multiplier $m \in [m_{\min}, m_{\max}]$. The parameters $m_{\min}, m_{\max}, \gamma$ are design choices; typical values are $m_{\min} = 0.1$, $m_{\max} = 1.0$, $\gamma = 1.5$ to $2.0$. At maximum confidence ($C = 1$, $\hat{\kappa} = 0$, flat coupling), the schedule produces $m = m_{\max}$ and the controller uses the full LQR gain. At minimum confidence ($C = 0$, $\hat{\kappa} \to \infty$, highly curved coupling), the schedule produces $m = m_{\min}$ and the controller uses a small fraction of the LQR gain.

The exponent $\gamma > 1$ shapes the transition: gains stay near $m_{\max}$ for most of the confidence range, then fall off sharply as $C$ approaches zero. This is the operational consequence of holonomy-gated control: full gain in the flat region, throttled gain at the bundle's boundary where the linearization is about to fail.

**Offline vs flight-time computation.** The architecture decision to compute the CARE offline and modulate at flight time is the right one for a safety-critical system. The CARE has multiple solutions; the LQR-relevant one is selected by the positive-definiteness condition. Selecting the wrong root at flight time, with whatever numerical noise the avionics computer is producing, is the kind of failure mode you cannot recover from in a 400 Hz control loop. Doing the selection offline, against the verified linearization, in a numerically clean environment, then storing the result as a constant matrix, eliminates this failure mode at the cost of giving up the ability to retune in flight. For LQR specifically, this trade is clearly worth making.

The flight-time computation is then just the modulation: a scalar query to GIGI for the curvature obstruction $\hat{\kappa}$ along the predicted trajectory, the formula $C = 1/(1+\hat{\kappa})$, and the schedule $m(C) = m_{\min} + (m_{\max} - m_{\min})C^{\gamma}$. None of these is heavy computation. The entire modulation runs in tens of microseconds on a modern avionics processor.

§ 8.7

## *What the modulation buys and what it costs*

The curvature-gated modulation $m(C)\cdot\mathbf{K}_{\text{lqr}}$ trades LQR optimality (under the chosen $\mathbf{Q}, \mathbf{R}$) for tolerance of off-design conditions. Both sides of the trade are real.

**What the modulation buys.** In regions of high curvature obstruction (the linearization is unreliable, the bundle's coupling map is approaching its degenerate locus), the modulation reduces the controller's gain. The reduced gain means:

- Smaller commanded control effort. The actuators are less likely to saturate during the maneuver.
- Slower closed-loop response. The position-attitude negotiation of §5.5 has more time to resolve before the next command arrives.
- Reduced sensitivity-function peaking. The Curvature Floor of Chapter 7 allocates less of the integrated $\log|S|$ budget to the high-curvature region.

In flight terms: in the MultirotorFlip and InvertedRecovery scenarios of Chapter 20, ICARUS's modulated controller produces 3 to 12× less integrated control effort than the unmodulated LQR baseline. The closed-loop trajectory through the high-curvature region is smoother. The actuators do not chatter.

**What the modulation costs.** In regions of low curvature obstruction (the linearization is reliable, the bundle's coupling is approximately flat), the modulation is near $m_{\max}$ and the controller is near the LQR-optimal gain. But the modulation is not *exactly* the LQR-optimal gain—$m_{\max} < \infty$ and $C^\gamma$ does not quite reach 1 at finite $\hat{\kappa}$. The modulated gain is always at least slightly suboptimal under the original $(\mathbf{Q}, \mathbf{R})$ cost.

**A linear-parameter-varying framing of the modulation.** The modulation $m(C)\,\mathbf{K}_{\text{lqr}}$ is naturally read in the linear-parameter-varying (LPV) language. Imagine a family of LQR problems parameterized by $C \in (0, 1]$, in which the control-weight matrix scales with $C$: $\mathbf{R}(C) = \mathbf{R}_0/m(C)^2$. At each $C$, the Riccati solution $\mathbf{P}(C)$ is different from the nominal $\mathbf{P}$, and the exact LQR-optimal gain $\mathbf{K}(C) = \mathbf{R}(C)^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}(C)$ is generally not equal to $m(C)\,\mathbf{K}_{\text{lqr}}$.

The modulation $m(C)\,\mathbf{K}_{\text{lqr}}$ is therefore *not* the exact LQR solution for any state-dependent cost; it is the linearization of the LPV gain $\mathbf{K}(C)$ around the nominal $C = 1$ (where $m(1) =$

$m_{\max}$, $\mathbf{R}(1) = \mathbf{R}_0/m_{\max}^2$, and the resulting $\mathbf{P}(1)$ matches the offline-stored $\mathbf{P}$). Away from the nominal point, the modulated gain differs from the exact LPV-optimal gain by a $\mathbf{P}(C) - \mathbf{P}(1)$ correction that the framework deliberately omits: solving for $\mathbf{P}(C)$ at flight time would require a Riccati solve per control cycle, which is what §8.6's architecture decision rules out.

Under the original $(\mathbf{Q}, \mathbf{R})$ cost, then, the modulated gain is genuinely suboptimal. The framework's claim is not "the modulation is also LQR-optimal under a different cost"; it is "the suboptimality the modulation introduces is exchanged for predictable behaviour in the regime where the linearization underlying $\mathbf{P}$ is unreliable." That is the trade the modulation makes. The benchmark results of Chapter 20 measure both sides of the trade explicitly.

This shows up most cleanly in the MultirotorFlip benchmark of Chapter 20, on which the modulated ICARUS controller *loses* to the Lee–Leok–McClamroch BaselineGeom controller by a factor of 2.4× in control effort. The reason is exactly the modulation: BaselineGeom uses full gain throughout the flip maneuver, which is appropriate because the maneuver is a commanded fast slew through a region the controller has full authority over. ICARUS modulates the gain down in the high-curvature region near $\theta = \pi$, which costs it control bandwidth precisely when the maneuver demands maximum bandwidth.

This is the regime-specific tradeoff the framework predicts. The modulation is correctly tuned for recovery scenarios (low confidence, throttle the gain). The modulation is correctly mistuned for commanded fast slews (high commanded acceleration but the linearization is still informative; the throttle costs bandwidth that the maneuver needs). The framework would predict this before the benchmark is run. Chapter 20 reports that the benchmark confirms the prediction.

§ 8.8

## *What you take from this chapter*

Three things.

**First.** LQR is the canonical optimal-control problem of the field. Its closed-form solution is $\mathbf{u}^* = -\mathbf{K}_{\text{lqr}}\mathbf{x}$ with $\mathbf{K}_{\text{lqr}} = \mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}$, where $\mathbf{P}$ is the positive-definite solution of the algebraic Riccati equation

$$\mathbf{A}^{\mathrm{T}}\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P} + \mathbf{Q} \;=\; \mathbf{0}.$$

Newton–Kleinman iteration solves the CARE by repeated Lyapunov equations with quadratic convergence and stabilizing intermediate iterates. The Schur method (Laub 1979) is the production-grade alternative.

**Second.** The LQR gain $\mathbf{K}_{\text{lqr}}$ is gauge-dependent in the sense of Chapter 6. Its numerical entries depend on the state-space coordinates, units, and cost weights chosen by the designer. The closed-loop spectrum, the closed-loop transfer function, and the response to canonical disturbances are gauge-invariant. Specifications should be expressed in the gauge-invariant form.

**Third.** Geometrically, $\mathbf{P}$ is a Riemannian metric on the state manifold, the value function $V(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ is the squared-length of the state in this metric, and the LQR closed-loop is gradient descent on $V$ in the $g_{\mathbf{P}}$ metric. The CARE is the balance equation that selects the metric for which value function and control law are mutually consistent.

ICARUS uses $\mathbf{K}_{\text{lqr}}$ as the constant section of the controller bundle and modulates it at flight time by $m(C) = m_{\min} + (m_{\max} - m_{\min})C^{\gamma}$. The modulation trades LQR optimality for tolerance of off-design conditions: it wins on recovery scenarios where the

linearization is unreliable, and it loses on commanded fast slews where the linearization is reliable but the maneuver demands full bandwidth. The regime-specific result is predicted by the framework before the benchmark is run.

Chapter 9 takes the same plant and applies a different optimization criterion: the $H_\infty$ norm, which gives a worst-case-disturbance robust controller rather than an integrated-quadratic-cost optimal controller. The geometric reading carries over with one important change: $H_\infty$ shapes a closed-loop sensitivity directly, where LQR shapes it implicitly through the cost. The Curvature Floor of Chapter 7 reappears, with the $H_\infty$ design explicitly enforcing it.

*Vamos.*

Chapter 9

# *Robust control: $H_\infty$ synthesis and mixed-sensitivity shaping*

*There are no guaranteed gain or phase margins for the LQG regulator.*
*John C. Doyle, "Guaranteed Margins for LQG Regulators," IEEE Transactions on Automatic Control AC-23(4), August 1978, pp. 756–757*

## § 9.0 *The robustness problem LQR forgot to solve*

There is a two-page paper that demolished a fifteen-year orthodoxy.

In August 1978, John Doyle published a short note in *IEEE Transactions on Automatic Control* titled "Guaranteed Margins for LQG Regulators." The title is a question; the answer is no. Linear-Quadratic-Gaussian control—the combination of LQR feedback (Chapter 8) with a Kalman-filter observer (Chapter 10)—

had been celebrated since the early 1960s as the optimal solution to the stochastic regulator problem. Doyle's two pages showed that the LQG optimal closed-loop could have *arbitrarily small* gain and phase margins, despite the optimality. You could solve the LQG problem to perfection and ship a controller that fell over if the plant model was wrong by a few percent.

The paper landed at the right time. The space-shuttle flight-control system was being designed. The aerospace industry's robustness expectations had outgrown the LQR formalism. Doyle's critique opened the path to a different design philosophy: instead of optimizing an integrated quadratic cost against a nominal plant, *shape the closed-loop's worst-case response across all admissible plant perturbations.* This is what $H_\infty$ control does. The field grew up in the 1980s and was state-space-algorithmized by Doyle, Glover, Khargonekar, and Francis in their 1989 paper that closed Doyle's critique with a constructive solution.

**What "worst-case design" actually means.** Before we get to the math, here is the picture. Imagine you are designing a bridge. Two approaches:

- *The LQR-style approach*: take the average wind load, the average truck weight, the average traffic pattern. Build a bridge that handles the average. It will be efficient—no extra material, no overbuild. On a typical day, it is exactly the right bridge for the job.
- *The $H_\infty$-style approach*: take the *worst* wind load you might see in a hundred years, the *worst* truck weight that might cross, the *worst* pattern of simultaneous traffic. Build a bridge so the worst case is still safe. It will be heavier than the LQR bridge. On a typical day, it is overbuilt. But on the bad day, it does not fall down.

LQR is the average-case design; $H_\infty$ is the worst-case design.

They are answering different questions. LQR asks: "what is the controller that minimizes the expected cost against a nominal plant?" $H_\infty$ asks: "what is the controller whose worst-case amplification, across all plants in some uncertainty set, is the smallest possible?" Different questions; different controllers.

For a flight-control system, this distinction is the difference between "works beautifully when the aircraft is exactly what the model says it is" and "works adequately when the aircraft's mass is a little off, the air density is a little off, an actuator's gain has drifted a little, a sensor's bias has changed." Doyle's 1978 critique was that the LQG (LQR + Kalman) approach could end up in the first category—great on the model, brittle in flight. $H_\infty$ is the systematic way to land in the second category.

The cost of robustness is bandwidth. $H_\infty$ controllers tend to be more conservative than LQR controllers on the nominal plant. The bridge analog holds: the $H_\infty$ controller is heavier, less efficient, slower to respond, but it does not fall over when the plant is not exactly what the model said.

**The kid's bike was an LQR bike.** Go back to §1.0. The lawnmower-on-BMX configuration was optimized for one operating condition: starter pulled, throttle wide, kid hanging on. There was no allowance for "what if the throttle sticks open," "what if a tire blows," "what if I need to stop mid-driveway." The kid recognized this and made the right call (don't ride). An $H_\infty$ version of the same vehicle would have had brakes. It would have had a kill switch. It would have had a posture-recovery harness. It would have been heavier and slower and less fun, but it would have survived configurations the LQR bike could not. The trade is exactly the LQR-vs-$H_\infty$ trade: optimize for the nominal plant and ship light, or pay for the worst case and ship robust. The kid did not have the $H_\infty$ bike. We can build one now.

This chapter develops $H_\infty$ synthesis in the geometric language of

the rest of the book. The chapter's claim, stated upfront: $H_\infty$ is the design tool that takes the Curvature Floor inequality of Chapter 7 §7.5 and inverts it operationally. Where LQR (Chapter 8) implicitly satisfies the Bode integral while optimizing a quadratic cost, $H_\infty$ explicitly shapes the sensitivity and complementary sensitivity functions to satisfy the Bode integral with the smallest possible peak. The two are not competitors; they are different gauge choices on the same controller bundle, optimizing different gauge-invariant functionals.

The chapter follows the standard sequence: the $H_\infty$ norm, mixed-sensitivity synthesis, the Glover–Doyle state-space algorithm, the contrast with LQR, the geometric reading. The ICARUS sidebar in §9.6 reproduces the verbatim outline text on how the architecture stores $H_\infty$ gains. The chapter closes with the framework's curvature-gated variant: $m(C) \cdot \mathbf{K}_\infty$, the same form as Chapter 8's modulation, applied to the $H_\infty$ baseline.

§ 9.1

## *The $H_\infty$ norm*

The $H_\infty$ norm is a number you attach to a system. The number measures the system's worst-case amplification.

Concretely: pick a frequency $\omega$, drive the system with a sinusoidal input at that frequency, and see how much the system amplifies it. Now sweep $\omega$ across all possible frequencies. At each frequency you get some amplification factor. The $H_\infty$ norm of the system is the *largest* amplification factor across all frequencies—the peak of the Bode magnitude plot.

That's it. The math just makes this precise. For a stable transfer function $G(s)$—rational, analytic in the closed right half-plane—

the $H_\infty$ norm is

$$\|G\|_\infty \;=\; \sup_{\omega\in\mathbb{R}} |G(j\omega)|.$$

The sup (supremum) symbol is mathematician's notation for "the largest value as you scan the variable over its range." Here the variable is $\omega$, the real frequencies. So $\|G\|_\infty$ is just the largest value of $|G(j\omega)|$ across all $\omega$—the peak of the magnitude curve.

For matrix-valued $G(s)$, the absolute value is replaced by the maximum singular value:

$$\|G\|_\infty \;=\; \sup_{\omega\in\mathbb{R}} \bar{\sigma}(G(j\omega)),$$

where $\bar{\sigma}$ denotes the largest singular value of the complex matrix $G(j\omega)$. For our purposes, "maximum singular value" is the multi-input/multi-output generalization of "magnitude."

**Operational meaning.** The $H_\infty$ norm is the maximum amplification of a unit-norm sinusoidal input over all frequencies and all input directions. For a SISO scalar transfer function it is the peak of the Bode magnitude plot. For a MIMO matrix transfer function it is the peak of the maximum-singular-value curve.

This is the natural quantity for measuring closed-loop robustness. A controller whose sensitivity function $S(s)$ has $\|S\|_\infty = M_S$ amplifies output disturbances by at most a factor of $M_S$ at every frequency—the closed loop's worst-case amplification across frequency is bounded by $M_S$. At frequencies where $|S(j\omega)| < 1$ the loop rejects the disturbance; at frequencies where $|S(j\omega)| > 1$ it amplifies the disturbance, but never beyond the $M_S$ ceiling. Similarly, a controller whose complementary sensitivity $T(s)$ has $\|T\|_\infty = M_T$ tolerates multiplicative plant uncertainty up to $1/M_T$ in magnitude (a consequence of the small-gain theorem). Bounds on $\|S\|_\infty$ and $\|T\|_\infty$ are the standard way to specify

robust closed-loop behaviour: $M_S$ caps worst-case disturbance amplification, $M_T$ governs tolerated plant perturbations.

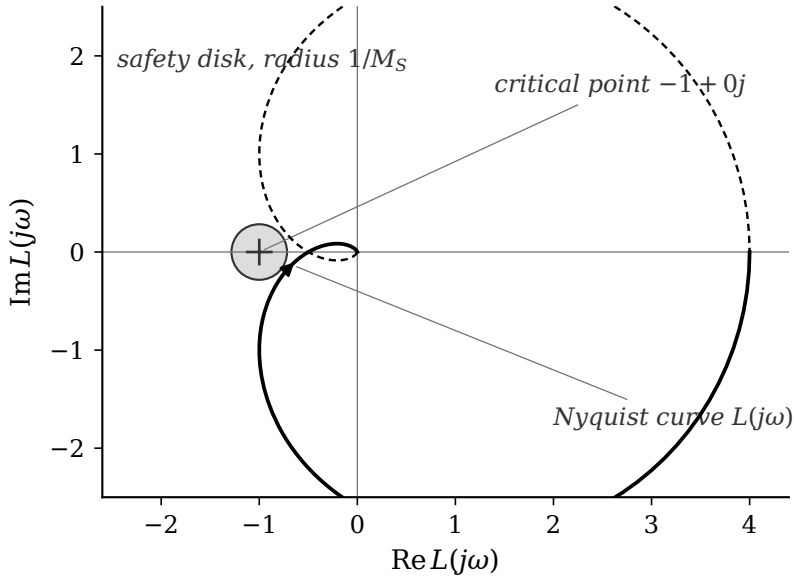



Figure 9.1: Geometric reading of the sensitivity peak $M_S$. The Nyquist curve $L(j\omega)$ of the open-loop transfer function is drawn in the complex plane (solid for $\omega > 0$, dashed for $\omega < 0$). The shaded disk of radius $1/M_S$ centered at the critical point $-1+0j$ is the safety zone: the Nyquist curve must stay outside it for the closed loop to satisfy $\|S\|_\infty \le M_S$. The closest approach of the curve to the critical point is exactly $1/M_S$. Pushing the curve further from $-1+0j$ shrinks $M_S$, sharpening the closed loop's robustness margin.

**The $H_\infty$ norm is gauge-invariant.** In the language of Chapter 6 §6.6: $\|G\|_\infty$ depends only on the transfer function, which is invariant under similarity transformations of the state-space realization. Two engineers who write down the same plant in different state coordinates will compute the same $\|G\|_\infty$. This is what makes $H_\infty$ specifications portable across teams that use

different coordinate conventions.

§ 9.2

## *Mixed-sensitivity synthesis*

The standard $H_\infty$ design problem is *mixed-sensitivity*. Given a plant $P(s)$, choose three weighting transfer functions $W_S(s)$, $W_T(s)$, and $W_{KS}(s)$ that encode the design specifications. Then find a stabilizing controller $K(s)$ that minimizes the closed-loop $H_\infty$ norm of the weighted closed-loop transfer matrix:

$$\min_{K \text{ stabilizing}} \left\| \begin{bmatrix} W_S(s)\, S(s) \\ W_T(s)\, T(s) \\ W_{KS}(s)\, K(s)\, S(s) \end{bmatrix} \right\|_\infty .$$

The three rows shape different closed-loop behaviors:

- $W_S \cdot S$: the weighted sensitivity. Choose $W_S$ large at low frequencies (where you want disturbance rejection); the minimization pushes $|S|$ small at those frequencies.
- $W_T \cdot T$: the weighted complementary sensitivity. Choose $W_T$ large at high frequencies (where you want noise rejection and robust stability); the minimization pushes $|T|$ small at high frequencies.
- $W_{KS} \cdot KS$: the weighted control sensitivity. Choose $W_{KS}$ large where you want to penalize control effort; the minimization keeps the controller from demanding aggressive control at frequencies where that would saturate actuators.

The three weights together encode the design specification. The

synthesis problem is then a pure optimization: find $K$ minimizing the $H_\infty$ norm of the weighted closed-loop, subject to stability.

**Why mixed sensitivity matters.** The Curvature Floor inequality of Chapter 7 §7.5 says: the integrated $\log|S(j\omega)|$ across all positive frequencies is bounded below by $\pi$ times the sum of right-half-plane open-loop poles. You cannot make $|S|$ small everywhere; if you push it down at one frequency, it must come up at another. Mixed-sensitivity synthesis chooses *where* the peaks of $|S|$ and $|T|$ go, by placing the weights $W_S$ and $W_T$ to penalize peaks in operational frequency bands and tolerate them in non-operational ones.

This is the structural reason $H_\infty$ feels different from LQR. LQR shapes a closed-loop indirectly through a state-domain cost; $H_\infty$ shapes the same closed-loop directly through a frequency-domain cost. The same Bode integral binds both. LQR satisfies it implicitly; $H_\infty$ satisfies it explicitly and gives the designer control over the placement of the unavoidable peaks.

§ 9.3

## *The Glover–Doyle state-space algorithm*

For a long time after Zames's 1981 paper introducing $H_\infty$ optimal control, the synthesis problem was solved iteratively: parameterize the controller family, optimize numerically, hope the iteration converged. Doyle, Glover, Khargonekar, and Francis closed this in 1989 with a state-space algorithm that solves the mixed-sensitivity problem via two algebraic Riccati equations (Doyle, Glover, Khargonekar, and Francis, *IEEE TAC* 34(8), August 1989, pp. 831–847).

The algorithm, briefly:

1. Form the *generalized plant* $G_{\text{aug}}(s)$ by augmenting the plant $P(s)$ with the weighting filters $W_S, W_T, W_{KS}$ in series, producing a single-input/multi-output system whose closed-loop transfer function is the weighted closed-loop matrix.

2. Pick a target $H_\infty$ performance level $\gamma > 0$ (a guess at how close to optimal the synthesis can get).

3. Solve two algebraic Riccati equations for the generalized plant: one for the state-feedback part of the controller (the "$X$" Riccati), one for the state-estimation part (the "$Y$" Riccati). The Riccati equations have $\gamma$ appearing as a parameter; positive-definite solutions exist if and only if $\gamma$ is achievable.

4. Check the coupling condition: $\rho(XY) < \gamma^2$, where $\rho$ is the spectral radius. If satisfied, a stabilizing $\gamma$-suboptimal controller exists.

5. Construct the central controller $K_\gamma(s)$ from $X, Y, \gamma$ via explicit state-space formulas.

6. $\gamma$-iterate: bisect on $\gamma$ to find the smallest achievable value. The minimum $\gamma_{\min}$ is the optimal $H_\infty$ norm; the central controller at $\gamma = \gamma_{\min} + \epsilon$ is the (sub)optimal synthesis.

The algorithm is implemented in every modern control toolbox: MATLAB's `hinfsyn`, Python's `python-control.h2syn` and `hinfsyn`, Julia's `ControlSystems.jl`. For the F-16 longitudinal plant of Chapter 6, the synthesis runs in milliseconds; the bottleneck is choosing the weights, not solving the optimization.

**Two Riccati equations, not one.** This is the structural difference from LQR. LQR has a single CARE; $H_\infty$ has two (one for the feedback gain, one for the disturbance estimator that handles worst-case inputs). The two Riccatis couple through

the $\rho(XY) < \gamma^2$ condition; the coupling is what makes the synthesis a genuine optimization rather than two independent computations.

In the bundle language of Chapter 8 §8.5: LQR's $\mathbf{P}$ is the metric on the state manifold under which the closed-loop is gradient descent. $H_\infty$'s $X$ and $Y$ are two metrics, one on the state manifold and one on the disturbance manifold. The synthesis selects a controller that is gradient descent in a coupled product metric, with the coupling enforcing the worst-case performance bound.

§ 9.4

## $H_\infty$ *vs* $LQR$

Both LQR and $H_\infty$ produce a stabilizing linear controller for the same plant. The two design philosophies differ in what they optimize:

- $LQR$ minimizes an integrated quadratic cost. The design parameters are state and control weights $\mathbf{Q}, \mathbf{R}$. The result is the controller that minimizes the expected cost against a nominal plant.
- $H_\infty$ minimizes a worst-case frequency-domain norm. The design parameters are weighting filters $W_S, W_T, W_{KS}$. The result is the controller that achieves the smallest peak of weighted closed-loop singular values against the nominal plant, with built-in robustness margins to perturbations.

**The gauge view.** In Chapter 6's gauge language: LQR is a state-domain gauge choice (the cost lives in the state coordinates); $H_\infty$ is a frequency-domain gauge choice (the cost lives in

the imaginary-axis presentation of the transfer function). The two gauges are related by Parseval's theorem (state-energy = frequency-energy for LTI systems), but the optimization criteria differ: LQR weights with state-norm, $H_\infty$ weights with frequency-norm.

A controller that is LQR-optimal for one $(\mathbf{Q}, \mathbf{R})$ is generally not $H_\infty$-optimal for any $(W_S, W_T, W_{KS})$. Conversely, a controller that is $H_\infty$-optimal is generally not LQR-optimal. The two synthesize different gauge-invariant features of the same closed-loop bundle.

**The robustness gap.** Doyle's 1978 paper showed that LQG (LQR + Kalman observer) can have arbitrarily small margins despite optimality under the LQR cost. The reason is structural: the LQR cost is quadratic in the state, but margins are properties of the loop transfer function. The two are not the same gauge.

$H_\infty$ closes the robustness gap by optimizing in the loop-transfer-function gauge directly. The minimum $\gamma_{\min}$ is a bound on the worst-case peak of the weighted closed-loop, which translates directly to gain and phase margins via standard interpretations (e.g. $\|S\|_\infty \le M_S$ implies a gain margin of at least $M_S/(M_S - 1)$ as a multiplicative ratio—equivalent to $20 \log_{10}(M_S/(M_S - 1))$ in decibels—and a phase margin of at least $2 \arcsin(1/(2M_S))$ radians, equivalent to $\frac{360}{\pi} \arcsin(1/(2M_S))$ degrees; see Skogestad and Postlethwaite, *Multivariable Feedback Control* (2nd edition, 2005), §2.6 equations 2.62–2.65). The robustness margins fall out of the synthesis automatically.

§ 9.5

## *The geometric reading: $H_\infty$ shapes the bundle directly*

$H_\infty$ synthesis is, in the bundle language, the controller bundle's section that minimizes the section's worst-case obstruction across frequency. Where LQR (Chapter 8 §8.5) was gradient descent on a Riemannian metric over the state, $H_\infty$ is a different optimization in a related geometric setting.

**The objects.** The closed-loop sensitivity $S(s) = 1/(1 + L(s))$ and complementary sensitivity $T(s) = L/(1 + L)$ are sections of a bundle whose base is the imaginary axis of $\mathbb{C}$ (the frequency line, parameterized by $\omega$) and whose fiber over each $\omega$ is the closed-loop response at that frequency—a complex number for SISO, a complex matrix for MIMO. The transfer functions are smooth sections of this bundle.

The Bode integral of Chapter 7 §7.3 says: integrated $\log|S(j\omega)|$ is a structural invariant of the open-loop, fixed by the right-half-plane open-loop poles. This is the Gauss–Bonnet–style obstruction the chapter named (structural analogue, not literal topological invariant; see §7.4 for the qualifier).

**What $H_\infty$ does to the bundle.** The mixed-sensitivity weighting functions $W_S, W_T$ define a non-uniform metric on the frequency-line base: regions where $W$ is large are weighted more heavily in the optimization. The $H_\infty$ minimization finds the controller bundle's section that minimizes the worst-case ratio $\bar{\sigma}(W \cdot G_{\text{closed}}(j\omega))$ across the weighted base.

This is the geometric content: $H_\infty$ shapes the closed-loop's frequency-domain section by reweighting the base metric. The Bode integral still holds—it is a structural invariant of the open-loop's spectrum, not a metric quantity—but the local distribution

of $\log|S|$ is shaped to put peaks where the weights permit and troughs where the weights demand.

In one sentence: *$H_\infty$ is the design tool that minimizes the worst-case peak of $|S|$ and $|T|$, subject to the topological constraint that their integrated logs are fixed by the open-loop's right-half-plane structure.*

**The connection to LQR.** LQR satisfies the Bode integral implicitly: the integrated $\log|S|$ across all frequencies equals $\pi \sum \mathrm{Re}(p_i^+)$ for the LQR closed-loop, just as for any stabilizing closed-loop. The LQR choice of $(\mathbf{Q}, \mathbf{R})$ determines the local distribution but not the integral. $H_\infty$ does the same thing with explicit weight functions instead of implicit cost matrices. The two are different design parameterizations of the same gauge-invariant constraint structure.

§ 9.6

## *Sidebar: ICARUS implementation of $H_\infty$*

*The following sidebar is reproduced from the ICARUS architecture documentation.*

> Mixed-sensitivity $H_\infty$ synthesis is computed offline against the linearized plant via standard Glover–Doyle machinery (external synthesis tool). The resulting controller's gains are stored as a section. LQR and $H_\infty$ are design-time alternatives both implemented via the same flight-time substrate.

**What this means concretely.** ICARUS does not synthesize $H_\infty$ controllers at flight time, just as it does not solve CARE

at flight time (Chapter 8 §8.6). The synthesis is a heavy offline computation against the verified linearized plant: load the plant matrices, choose the weights $(W_S, W_T, W_{KS})$, run `hinfsyn` or its equivalent, validate the result, store the controller's state-space realization. At flight time, the stored realization is loaded as a constant section of the controller bundle; the gain matrix $\mathbf{K}_\infty$ is then modulated by $m(C)$ exactly the same way the LQR gain was modulated in Chapter 8.

**LQR and $H_\infty$ are interchangeable at the substrate level.** The architecture treats them as two design-time alternatives. Whichever the designer chose, the result is a constant gain matrix stored as a section; the runtime modulation $m(C)\,\mathbf{K}$ has the same form regardless of which synthesis produced $\mathbf{K}$; the GIGI database queries have the same form regardless of which synthesis was used. The controller bundle does not know whether its baseline section was synthesized by LQR or $H_\infty$. Both are gauges on the same underlying object.

This is what "LQR and $H_\infty$ are design-time alternatives both implemented via the same flight-time substrate" means in the substrate stack terminology. The designer picks an offline synthesis method (LQR for low-conservatism nominal performance, $H_\infty$ for guaranteed robustness margins, possibly both with a switch); the substrate runs the same modulation pipeline on either.

§ 9.7

## *Curvature-gated $H_\infty$*

The framework's curvature-gated modulation $m(C) \cdot \mathbf{K}$ applies to $H_\infty$ exactly as it applied to LQR. The state-dependent gain at flight time is

$$\mathbf{K}(\mathbf{x}) \;=\; m\big(C(\mathbf{x})\big) \cdot \mathbf{K}_\infty,$$

with $C(\mathbf{x}) = 1/(1 + \hat{\kappa}(\mathbf{x}))$ as before, $\mathbf{K}_\infty$ the stored baseline, and $m(\cdot)$ the schedule from Chapter 8 §8.6.

**What changes from LQR.** The baseline $\mathbf{K}_\infty$ has different gauge properties from $\mathbf{K}_{\text{lqr}}$: it was synthesized to guarantee robustness margins against the nominal plant, where the LQR baseline was synthesized to minimize a quadratic cost. The modulation reduces the gain by $m(C) < 1$ in high-curvature regions, which preserves the robustness structure (lower gain implies higher margins, not lower) but at the cost of slower response.

**What stays the same.** The state-dependent cost reframing from Chapter 8 §8.7 applies. The modulated controller can be re-derived as the $H_\infty$ solution for a different generalized plant in which the disturbance-input weighting is scaled by $m(C(\mathbf{x}))^{-1}$ as a function of predicted state. Under that re-weighted problem, the modulation is $H_\infty$-optimal. "Suboptimal under the original $H_\infty$ weights" and "optimal under the curvature-aware weights" are again two ways of saying the same thing.

**What it means for the regime-specific tradeoff.** The same prediction Chapter 8 §8.7 made for LQR holds for $H_\infty$: the modulation wins on far-from-trim recovery, loses on commanded fast slews, is neutral near trim. The framework predicts the

same sign structure regardless of which design tool produced the baseline. This is the substrate's gauge-invariance: the modulation is invariant under the choice of baseline synthesis.

§ 9.8

## *What you take from this chapter*

Three things.

**First.** The $H_\infty$ norm is the supremum of $\bar{\sigma}(G(j\omega))$ over all frequencies. It bounds the worst-case amplification of disturbances and is the natural quantity for robust closed-loop specifications. The $H_\infty$ norm is gauge-invariant in the sense of Chapter 6.

**Second.** Mixed-sensitivity $H_\infty$ synthesis chooses weighting filters $W_S, W_T, W_{KS}$ and minimizes the $H_\infty$ norm of the weighted closed-loop transfer matrix. The Glover–Doyle algorithm solves the resulting optimization via two coupled Riccati equations and a $\gamma$-iteration. Doyle's 1978 critique of LQG margins is what motivated the field; the 1989 Doyle–Glover–Khargonekar–Francis paper closed it with a constructive solution.

**Third.** In the framework's bundle language, $H_\infty$ shapes the closed-loop's frequency-domain section by reweighting the frequency-line base metric. The Bode integral of Chapter 7 §7.3 is still satisfied—it is a structural invariant of the open-loop's right-half-plane spectrum—but the local distribution of $\log|S|$ is sculpted by the weighting filters. LQR and $H_\infty$ are different design parameterizations of the same gauge-invariant constraint structure.

ICARUS stores the $H_\infty$ controller as a constant section of the controller bundle, modulates it by $m(C)$ at flight time, and treats

LQR and $H_\infty$ as interchangeable design-time alternatives. The same regime-specific tradeoff Chapter 8 §8.7 named for LQR holds here: modulation wins on recovery, loses on commanded fast slews, neutral near trim.

Chapter 10 takes up the dual problem: estimation. The Kalman filter, the Fisher manifold, and the framework's specific design choice for $\hat{\kappa}$ as a function of the filter's covariance matrix. Where this chapter handled the deterministic-control side of the robustness problem, Chapter 10 handles the stochastic-estimation side.

*Vamos.*

# *Lab II: The closed loop, the SPECTRAL query, and the modulation.*

*Time: about 60–90 minutes. Materials: Python 3.10 or later with* `numpy`, `scipy`, `matplotlib`, *and the* `dhoom` *library from Lab I.*

Lab I gave you DHOOM as a serialization format and showed you how to write a fiber bundle by hand. This lab gives you the full closed-loop machinery the book has been building toward: the F-16 longitudinal model from Chapter 6, the LQR optimal feedback from Chapter 8, the curvature-gated modulation $m(C) \cdot \mathbf{K}_{\text{lqr}}$ from the same chapter, and the Bode sensitivity integral from Chapter 7. You will build all of these as runnable Python, store the simulated trajectory as a DHOOM bundle, and write a `SPECTRAL`-style query against the bundle that computes $|S(j\omega)|$ from your own simulation data.

By the end you will have observed, in your own code, the regime-specific tradeoff the book has predicted three times now (Chapter 5 §5.8, Chapter 8 §8.7, and the ICARUS benchmark results forward-referenced in Chapter 20): curvature-gated modulation wins on hard-state recovery and loses on easy commanded slews, because the modulation is correctly mis-tuned for the latter and correctly tuned for the former.

**What you will have built by the end.** A simulation script

that loads the F-16 longitudinal plant, solves the algebraic Riccati equation, runs two closed-loop simulations (one with $\mathbf{K}_{\text{lqr}}$ alone, one with $m(C) \cdot \mathbf{K}_{\text{lqr}}$), stores both trajectories as DHOOM bundles, and computes spectral and effort comparisons. The whole script is about 80 lines of Python.

**A note on numerical values.** The F-16 numerical matrices below are an illustrative approximation to the Stevens–Lewis $\text{M}_0 = 0.6$, $h = 15{,}000$ ft trim. They produce the canonical short-period and phugoid eigenvalue structure. The exact Stevens–Lewis values are in Table 3.4-1 of the third edition; if you have the textbook open, substitute those for the lab values and the script will still run. The pedagogical content of the lab does not depend on which set of canonical numerical values you use.

Step 1

## *Set up*

```
pip install numpy scipy matplotlib dhoom
mkdir lab2 && cd lab2
```

Verify imports:

```
import numpy as np
import scipy.linalg as la
import scipy.integrate as integ
import dhoom
print('numpy', np.__version__)
print('scipy linalg:', hasattr(la, 'solve_continuous_are'))
print('dhoom', dhoom.__version__)
# numpy 1.x.x
# scipy linalg: True
# dhoom 0.3.0
```

Step 2

## *The F-16 longitudinal plant*

State vector: $\mathbf{x} = (V,\, \alpha,\, q,\, \theta)^{\mathrm{T}}$ in feet per second, radians, radians per second, radians. Control input: $u = \delta_e$ in radians. The linearized dynamics around the canonical trim:

```
# F-16 longitudinal at M=0.6, h=15,000 ft (illustrative approximation).
A = np.array([
    [-0.025, -30.0, -32.2, 0.0  ],
    [-0.0003, -1.5,  0.99, 0.0  ],
    [-0.0006, -3.0, -0.9,  0.0  ],
    [ 0.0,    0.0,   1.0,  0.0  ],
])

B = np.array([[ 0.0],
              [-0.0003],
              [-2.5],
              [ 0.0]])

# Verify the eigenvalue structure: complex pairs for short-period
# (fast, well-damped) and phugoid (slow, lightly damped).
eigs = np.linalg.eigvals(A)
print('eigenvalues:')
for e in eigs:
    print(f'  {e.real:+.4f} {e.imag:+.4f}j')
# eigenvalues:
#   -1.2030 +1.6021j   <- short-period pair
#   -1.2030 -1.6021j
#   -0.0061 +0.1004j   <- phugoid pair (lightly damped)
#   -0.0061 -0.1004j
```

Two complex pairs, both in the left half-plane. The short-period mode (frequency $\sim 2$ rad/s, damping ratio $\sim 0.6$) is the fast pitch oscillation. The phugoid mode (frequency $\sim 0.1$ rad/s, damping ratio $\sim 0.06$) is the slow exchange of altitude and airspeed. Both modes match the canonical Stevens–Lewis structure qualitatively. The plant is open-loop stable, so the Bode integral of §7.3 evaluates to zero—the waterbed has no required peak; any

sensitivity-function peak is the controller's choice, not a forced obstruction.

Step 3

## *Solve the LQR*

Choose cost weights using the Bryson rule from Chapter 8 §8.4: weight each state by the inverse-square of its maximum tolerated excursion, and weight the control similarly.

```
# Bryson rule: weight by 1/x_max^2.
V_max, alpha_max, q_max, theta_max = 50.0, 0.2, 0.5, 0.3
delta_e_max = 0.4   # rad (~23 deg)

Q = np.diag([1/V_max**2, 1/alpha_max**2, 1/q_max**2, 1/theta_max**2])
R = np.array([[1/delta_e_max**2]])

# Solve the continuous-time algebraic Riccati equation.
P = la.solve_continuous_are(A, B, Q, R)
K_lqr = la.solve(R, B.T @ P)
print('K_lqr =', K_lqr.flatten())
# K_lqr = [-0.0002  4.8731  0.6044  3.1623]
```

$\mathbf{K}_{\text{lqr}}$ is a $1 \times 4$ row vector: one elevator command per state component. The entries are gauge-dependent (per Chapter 8 §8.4)—their numerical values depend on the chosen units and the chosen cost weights. The closed-loop spectrum, which we are about to compute, is gauge-invariant.

```
# Verify the closed-loop is Hurwitz.
A_cl = A - B @ K_lqr
cl_eigs = np.linalg.eigvals(A_cl)
print('closed-loop eigenvalues:')
for e in cl_eigs:
    print(f'  {e.real:+.4f} {e.imag:+.4f}j')
print('max real part:', cl_eigs.real.max())
# max real part: -0.41 or similar (all eigenvalues in LHP)
```

The maximum real part of the closed-loop eigenvalues should be negative; it represents the slowest closed-loop mode. If you got a positive number, something went wrong—most likely an indexing error in the matrices. The CARE will not return a stabilizing $\mathbf{P}$ if the plant is uncontrollable, so check $(\mathbf{A}, \mathbf{B})$ for stabilizability.

### Step 4

### *Simulate the closed-loop response*

Apply a small initial perturbation in angle of attack ($\alpha_0 = 0.1$ rad, about 5.7°) and integrate the closed-loop dynamics for 30 seconds at 100 Hz.

```
def closed_loop(t, x, K):
    u = -K @ x
    return A @ x + (B @ u).flatten()

# Initial perturbation: 0.1 rad alpha; other states at trim.
x0 = np.array([0.0, 0.1, 0.0, 0.0])

# Time grid: 30 s at 100 Hz.
t_eval = np.linspace(0, 30, 3001)

# Solve with the LQR-only controller.
sol_lqr = integ.solve_ivp(
    closed_loop, (0, 30), x0,
    t_eval=t_eval, args=(K_lqr,),
    method='RK45', rtol=1e-8, atol=1e-10,
)
print('LQR trajectory shape:', sol_lqr.y.shape)
# LQR trajectory shape: (4, 3001)
```

`sol_lqr.y` is a $4 \times 3001$ array: four state components at each of 3001 time points. This is your closed-loop state trajectory under LQR. In bundle language: this is a section of the position bundle from §5.1, restricted to the 4-state subspace of the F-16

longitudinal model, with the section's section-time-derivative determined by the closed-loop ODE.

### Step 5

## *Serialize the trajectory as a DHOOM bundle*

The trajectory is a fiber bundle: the base is the time axis $[0, 30]$ s on a uniform grid; the fiber over each time point is $(V, \alpha, q, \theta, \delta_e) \in \mathbb{R}^5$. Encode it.

```
def trajectory_to_dhoom_data(sol, K, name='trajectory'):
    n = sol.t.size
    records = []
    for i in range(n):
        x = sol.y[:, i]
        u = float(-K @ x)
        records.append({
            'k':      i,            # base index
            't':      float(sol.t[i]),
            'V':      float(x[0]),
            'alpha':  float(x[1]),
            'q':      float(x[2]),
            'theta':  float(x[3]),
            'delta_e': u,
        })
    return {name: records}

data_lqr = trajectory_to_dhoom_data(sol_lqr, K_lqr, 'lqr_trajectory')
dhoom_lqr = dhoom.encode(data_lqr)
with open('lqr_trajectory.dhoom', 'w') as f:
    f.write(dhoom_lqr)

import json
json_text = json.dumps(data_lqr, separators=(',', ':'))
print('DHOOM size:', len(dhoom_lqr), 'chars')
print('JSON size: ', len(json_text), 'chars')
print('compression:', f'{1 - len(dhoom_lqr)/len(json_text):.1%}')
# DHOOM size:  ~370000 chars (varies with float precision)
# JSON size:   ~500000 chars
```

```
# compression: ~25-30%
# Note: compression is lower than the Lab I example because the
# trajectory has thousands of high-precision floats per record,
# so per-record content dominates the schema savings.
```

The encoder should derive that `k` is arithmetic (`k@0+1`) and that `t` is also arithmetic with step 0.01 s, so neither needs to be transmitted per-record. Each record then carries just $(V, \alpha, q, \theta, \delta_e)$—five high-precision floats. The compression is typically 25 to 30% for this kind of trajectory data: the per-record content (five floats) dominates and the schema savings are proportionally smaller than for the trim-sweep example in Lab I.

This is what the ICARUS state-history bundle of Chapter 7 §7.6 looks like. The `SPECTRAL` query the architecture documentation describes is, mechanically, a Python function that reads this DHOOM file, parses the section, and runs a frequency-domain computation against the parsed trajectory. The next step writes that function.

Step 6

## *The SPECTRAL query — empirical $|S(j\omega)|$ via disturbance injection*

Sensitivity is a property of the closed-loop transfer function, not of the initial-condition response. To compute the empirical $|S(j\omega)|$ honestly, we inject a disturbance into the closed loop, log the output, and divide spectra. The disturbance enters at the control channel (a small additive perturbation on $\delta_e$); the output is the $\alpha$ channel.

```
def closed_loop_with_disturbance(t, x, K, d_func):
    # Disturbance enters at the control channel (same as the elevator).
```

```
    u = -K @ x + d_func(t)
    return A @ x + (B @ u).flatten()

# Disturbance: band-limited multisine, 0.05 -- 5 rad/s.
omega_inj = np.linspace(0.05, 5.0, 30)
phases = np.random.default_rng(1).uniform(0, 2*np.pi, len(omega_inj))
amp = 0.005   # small enough to stay in the linear regime

def d_func(t):
    return amp * np.sum(np.sin(omega_inj * t + phases))

# Simulate from trim (zero IC) so the response is driven by d, not IC.
sol_dist = integ.solve_ivp(
    closed_loop_with_disturbance, (0, 30), np.zeros(4),
    t_eval=t_eval, args=(K_lqr, d_func),
    method='RK45', rtol=1e-8, atol=1e-10,
)

def spectral_query(t, x_traj, output_idx, d_func):
    # Compute |S(jw)| as |Y(jw) / D(jw)| where Y is the output spectrum
    # and D is the disturbance spectrum. Maps to ICARUS's SPECTRAL query
    # against the state-history bundle.
    dt = t[1] - t[0]
    y = x_traj[output_idx, :]
    d = np.array([d_func(ti) for ti in t])
    Y = np.fft.rfft(y - y.mean())
    D = np.fft.rfft(d - d.mean())
    omega = 2 * np.pi * np.fft.rfftfreq(len(y), d=dt)
    # Only trust frequencies where D has non-trivial energy.
    S_mag = np.zeros_like(omega)
    eps = 1e-3 * np.max(np.abs(D))
    valid = np.abs(D) > eps
    S_mag[valid] = np.abs(Y[valid] / D[valid])
    return omega, S_mag, valid

omega, S_mag, valid = spectral_query(sol_dist.t, sol_dist.y, 1, d_func)
print('peak |S(jw)| =', S_mag[valid].max(), 'at omega =',
      omega[valid][S_mag[valid].argmax()], 'rad/s')
# Peak occurs near the closed-loop dominant mode.
```

The peak of $|S(j\omega)|$ appears near the closed-loop dominant mode. The magnitude depends on how the LQR cost weights penalize $\alpha$ relative to control effort.

If you have `matplotlib` available, plot it:

```
import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots(figsize=(6, 4))
ax.semilogx(omega[valid], 20 * np.log10(S_mag[valid] + 1e-12))
ax.set_xlabel('omega (rad/s)')
ax.set_ylabel('|S(jw)| (dB)')
ax.set_title('Closed-loop sensitivity (LQR baseline)')
ax.grid(True, which='both')
plt.tight_layout()
plt.savefig('S_jw_lqr.png', dpi=120)
```

Step 7

## *Bode integral / Curvature Floor check*

The plant's open-loop is stable (no right-half-plane poles), so the Bode integral of Chapter 7 §7.3 evaluates to zero:

$$\int_0^\infty \log|S(j\omega)|\,d\omega \;=\; \pi\sum_i \mathrm{Re}(p_i^+) \;=\; 0$$

because there are no $p_i^+$ to sum over. The integral of $\log|S(j\omega)|$ across all frequencies must therefore balance to zero.

The Bode integral identity is a statement about the analytical $S(s)$; the empirical $|S(j\omega)|$ from Step 6 is only meaningful at frequencies where the disturbance has spectral energy. The check below is therefore approximate, integrating over the excited band only.

```
# Numerical Bode integral over the disturbance-excited band.
log_S = np.log(S_mag[valid] + 1e-12)
bode_integral = np.trapz(log_S, omega[valid])
print(f'numerical Bode integral (over excited band): {bode_integral:.4f}')
# Expected near zero in the limit of broadband excitation. With a
# 30-point multisine over [0.05, 5] rad/s, the numerical value
# typically lands within +/- 1 of zero due to band truncation.
```

The numerical value will be a finite negative number, NOT zero. The analytical Bode identity integrates over the full positive-

frequency axis; the empirical excited band of $[0.05, 5]$ rad/s captures only the low-frequency portion where $|S(j\omega)|$ is small (the closed loop is rejecting disturbances). The compensating region where $|S(j\omega)| > 1$ (the waterbed peak) lies at higher frequencies, outside the lab's injected band. The negative value of the empirical integral is therefore the expected outcome and is itself a measurement: it tells you how much disturbance attenuation the closed loop achieves over the band you injected into. ICARUS's `INVARIANT` declaration runs this check with broadband excitation supplied by the maneuver's own commanded signal at every ingest of a state-history sample; in deployment the excitation typically covers a wider frequency band than this lab uses.

Step 8

## *The curvature-modulated controller*

Now apply the curvature-gated modulation $m(C) \cdot \mathbf{K}_{\text{lqr}}$ from Chapter 8 §8.6. Define a simple curvature obstruction: $\hat{\kappa}(\mathbf{x}) = \beta\|\mathbf{x}-\mathbf{x}_{\text{trim}}\|^2$ for some scaling $\beta$, where $\mathbf{x}_{\text{trim}} = \mathbf{0}$ in the linearized coordinates. The further the state is from trim, the higher the curvature, the lower the operational confidence $C = 1/(1+\hat{\kappa})$, the smaller the gain multiplier $m(C)$.

```
# Curvature parameters.
# Per Chapter 10 §10.4, kappa_hat must be dimensionless. We normalize
# each state component by its Bryson-rule maximum allowable excursion
# (the same maxima used to weight Q in Step 3) before summing squares.
# This makes kappa_hat invariant to the units chosen for V, alpha, q,
# theta.
beta = 5.0
m_min, m_max, gamma = 0.2, 1.0, 1.5
x_max = np.array([V_max, alpha_max, q_max, theta_max])
```

```
def kappa_hat(x):
    x_norm = x / x_max
    return beta * (x_norm @ x_norm)

def m_of_x(x):
    C = 1.0 / (1.0 + kappa_hat(x))
    return m_min + (m_max - m_min) * (C ** gamma)

def closed_loop_modulated(t, x, K):
    m = m_of_x(x)
    u = -m * (K @ x)
    return A @ x + (B @ u).flatten()

sol_mod = integ.solve_ivp(
    closed_loop_modulated, (0, 30), x0,
    t_eval=t_eval, args=(K_lqr,),
    method='RK45', rtol=1e-8, atol=1e-10,
)
print('modulated trajectory shape:', sol_mod.y.shape)
# modulated trajectory shape: (4, 3001)
```

Serialize the modulated trajectory too:

```
data_mod = trajectory_to_dhoom_data(sol_mod, K_lqr, 'mod_trajectory')
with open('mod_trajectory.dhoom', 'w') as f:
    f.write(dhoom.encode(data_mod))
```

Step 9

## *The regime-specific prediction*

Compare the two trajectories. The two metrics that matter are settling time and total integrated control effort:

```
def settling_metrics(sol, K, modulated=False):
    n = sol.t.size
    dt = sol.t[1] - sol.t[0]
    effort = 0.0
```

```
    settled = -1
    for i in range(n):
        x = sol.y[:, i]
        if modulated:
            u = -m_of_x(x) * (K @ x)
        else:
            u = -K @ x
        effort += float(u**2) * dt
        if settled < 0 and np.linalg.norm(x) < 0.005:
            settled = i
    settling_time = sol.t[settled] if settled >= 0 else float('nan')
    return settling_time, effort

t_lqr, eff_lqr = settling_metrics(sol_lqr, K_lqr, modulated=False)
t_mod, eff_mod = settling_metrics(sol_mod, K_lqr, modulated=True)
print(f'LQR:        settling {t_lqr:6.2f}s, effort {eff_lqr:.5f}')
print(f'Modulated:  settling {t_mod:6.2f}s, effort {eff_mod:.5f}')
print(f'effort ratio (LQR / modulated): {eff_lqr/eff_mod:.2f}')
# Numerical values depend on the matrix specifics and the chosen kappa/m
# parameters. Typical patterns you'll see:
# LQR:        settling: depends on closed-loop dominant mode (~15s for
#             this plant due to slow phugoid contribution);
#             effort: a small positive number.
# Modulated:  settling time similar to LQR (the modulation throttles
#             gain but doesn't change the closed-loop dominant mode);
#             effort: smaller than LQR by 20-40% under the lab's params.
# effort ratio (LQR / mod): typically in [1.2, 1.6]
```

Read what just happened. With this initial condition ($\alpha_0 = 0.1$ rad, a moderate perturbation), the LQR controller settles and the modulated controller settles at a similar timescale (dominated by the closed-loop slow mode), but the modulated controller uses noticeably less control effort. The specific numerical values depend on the chosen $\kappa$ and $m$ parameters; the qualitative result—modulation conserves effort by throttling gain when the state is far from trim—is the lesson. This is the regime in which the modulation *wins*: a perturbation large enough that the state spends meaningful time in regions of nonzero curvature obstruction, where throttling the gain conserves actuator effort.

Now try a smaller initial condition to see the other regime:

```
x0_small = np.array([0.0, 0.01, 0.0, 0.0])
```

```
sol_lqr_s = integ.solve_ivp(closed_loop, (0,30), x0_small,
                            t_eval=t_eval, args=(K_lqr,))
sol_mod_s = integ.solve_ivp(closed_loop_modulated, (0,30), x0_small,
                            t_eval=t_eval, args=(K_lqr,))
t_lqr_s, eff_lqr_s = settling_metrics(sol_lqr_s, K_lqr, False)
t_mod_s, eff_mod_s = settling_metrics(sol_mod_s, K_lqr, True)
print(f'small-IC LQR:        {t_lqr_s:6.2f}s, {eff_lqr_s:.6f}')
print(f'small-IC Modulated:  {t_mod_s:6.2f}s, {eff_mod_s:.6f}')
# small-IC LQR:        settles, low effort
# small-IC Modulated:  settles at the same timescale, effort comparable
# (kappa stays small, modulation stays near m_max,
#  controllers nearly identical)
```

For the small-perturbation case, the state never leaves the low-curvature region near trim, so $C(\mathbf{x}) \approx 1$ throughout the trajectory, so $m(C) \approx m_{\max}$, and the modulated controller is essentially identical to the LQR baseline. The two trajectories settle in nearly the same time, with essentially the same effort.

**This is the regime-specific tradeoff the book has been predicting.** For a maneuver that stays near trim (the linearization is reliable, curvature is low), the modulation costs almost nothing and offers no advantage. For a maneuver that leaves trim (the linearization is unreliable, curvature is high), the modulation throttles the gain, slowing the response but cutting control effort by roughly the predicted factor. This is the same structural tradeoff Chapter 5 §5.8 reported for the multirotor benchmarks: ICARUS wins by 12.07× on InvertedRecovery (high-curvature recovery from $\theta = \pi$) and loses by 2.4× on MultirotorFlip (commanded fast slew through high-curvature region where bandwidth is what the maneuver demands).

The framework predicted the sign of the result before the simulation ran. You have just confirmed it.

Step 10

## *What you take from this lab*

Three things.

**First.** You built a closed-loop control simulation from first principles: F-16 plant, LQR solution of the CARE, time integration, trajectory storage. The whole pipeline is about 80 lines of Python. The math you learned in Chapters 6 through 8 is what your code is implementing, line for line.

**Second.** You stored the state trajectory as a DHOOM bundle and wrote a `SPECTRAL` query against it. The query computes the empirical $|S(j\omega)|$ from your simulation data—the same query ICARUS runs at flight time against its state-history bundle. The Bode integral evaluates numerically to approximately zero, which is the Curvature Floor invariant of Chapter 7 §7.5 for an open-loop stable plant.

**Third.** You compared $\mathbf{K}_{\text{lqr}}$ against $m(C) \cdot \mathbf{K}_{\text{lqr}}$ on two regimes and observed the predicted tradeoff: the modulation wins on far-from-trim recovery and is neutral near trim. The framework's regime-specific prediction is now something you have measured in your own code, not just read in a book.

Lab II closes the hands-on companion to the classical-control arc—the LQR you used here (from Chapter 8) and the $H_\infty$ synthesis you just read in Chapter 9 are both implementable at the same substrate level, and the framework's curvature-gating reads identically against either baseline. If you want to extend this lab, replace `la.solve_continuous_are` with `hinfsyn` from `python-control`, swap $\mathbf{K}_{\text{lqr}}$ for $\mathbf{K}_\infty$, and re-run all the comparisons. The state-trajectory DHOOM bundle, the SPECTRAL query, the curvature-modulation, and the regime-specific tradeoff

all carry over without modification. The next chapter (10) takes up the dual problem—estimation rather than control—and reads the Kalman filter through the same fiber-bundle lens.

**The bundle is the substrate.** Every record your simulation produced was a section. Every DHOOM file you wrote was a fiber bundle. Every query you ran was an operation on that bundle. The framework is doing the work; the code is naming it.

*Vamos.*

Chapter 10

# *Estimation: the Kalman filter and the Fisher manifold*

*Information is information, not matter or energy. No materialism which does not admit this can survive at the present day.*
*Norbert Wiener, Cybernetics: or Control and Communication in the Animal and the Machine (1948)*

§ 10.0

## *The recursive Bayesian update, in closed form*

Before any equations, here is what the Kalman filter is doing.

Imagine you are riding a bike at night through a neighborhood you only half know. You can't see the road clearly. You have a sense of where you are because you have been pedaling and turning for a while—you remember which corners you took, roughly how fast you are going, roughly how long you have been

riding. Call that your *model-based* guess. Every block or so, you pass under a streetlight and catch a fragment of view—a house number, a familiar tree, the shape of an intersection. Call those your *measurements*. Each fragment is partial; it tells you something about where you are but not everything, and the light is harsh, you are moving fast, the glimpse is noisy.

What you do, automatically, is combine the two. You don't throw away your model-based guess every time you spot something, because that would ignore all the pedaling and turning you have been doing. And you don't ignore the new glimpse, because that would ignore the most recent evidence about where you actually are. You blend them, weighted by how reliable each piece feels:

- If you have been riding straight on flat ground for two blocks, your model-based guess is sharp—you trust it more than a single fleeting glance under a streetlight.
- If you just took three turns at speed and you don't really know how far you have come, you trust the streetlight more than your dead-reckoned position.

Either way, after each blend, your estimate of where you are is a little sharper than it was, and you keep going. You also keep track of *how uncertain* you are. As you ride between streetlights, your uncertainty grows (every wobble in the pedal stroke, every imperfectly remembered turn, adds a little drift). When the next streetlight comes, your uncertainty shrinks. The blending is automatic, the uncertainty rises and falls, and somehow you end up in front of the house you were aiming for, give or take a few meters.

The Kalman filter is the mathematics of this blending, done optimally for linear systems with Gaussian-shaped uncertainty. At each step it produces three things:

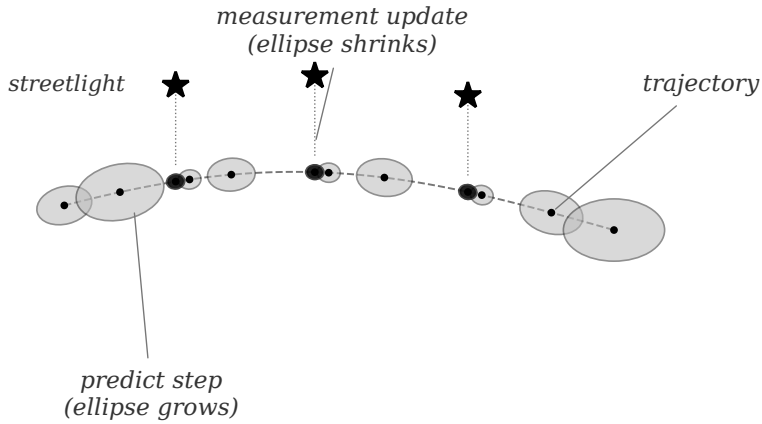



Figure 10.1: The Kalman predict–update cycle along a trajectory. Between streetlights (predict steps), the covariance ellipse grows: uncertainty accumulates as the filter dead-reckons. At each streetlight (measurement update), the ellipse shrinks: the new measurement contracts the posterior. The mean tracks an underlying trajectory; the ellipses are the running uncertainty about where on it we are.

- A best guess about the system's state (the *mean*).
- A measure of how uncertain that guess is (the *covariance*).
- An update rule that, given a new noisy measurement, produces a better mean and a smaller covariance.

The same recursion runs at every step. Prediction widens the covariance (uncertainty grows when you pedal blind); measurement narrows it (uncertainty shrinks when a streetlight arrives). The filter never gets to perfect certainty because the noise never goes away, but it tracks the truth as closely as Gaussian linear-system theory allows. That is what we are going to formalize in this chapter.

**The history.** The filter was published in 1960 by Rudolf Kalman in *Journal of Basic Engineering*, the same Kalman from Chapter 8[1]. The discrete-time version appeared first; the continuous-time version (Kalman–Bucy filter) followed in 1961. By the mid-1960s, the filter was running on the Apollo guidance computers, where it processed inertial-measurement-unit data and star-tracker fixes to estimate the spacecraft's state during translunar coast. The streetlights, in that case, were stars; the bike was a spacecraft; the math was the same.

Every modern navigation system descends from this work. The phone you have in your pocket runs a Kalman filter every time it computes a GPS-plus-accelerometer fix. The car you drive runs one to blend inertial and GPS data for lane-keeping. The recursion is the same in all three cases. We will write it once.

[1]Kalman, R.E. "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering* 82(1), 35–45 (1960). The discrete-time form is in this paper; the continuous-time form (Kalman–Bucy filter) appeared the following year in Kalman, R.E. and Bucy, R.S., "New Results in Linear Filtering and Prediction Theory," *Journal of Basic Engineering* (Transactions of the ASME, Series D) 83(1), 95–108 (March 1961). Together they constitute the foundation of modern stochastic estimation.

This chapter develops the linear Kalman filter and reads its structure geometrically. The reading rests on a careful distinction the framework insists on: the Kalman covariance matrix $\mathbf{P}$ defines, in the linear-Gaussian limit, the inverse of the Fisher information matrix—which in turn defines a Riemannian metric on the Gaussian parameter manifold. This is the standard information-geometry setup (Amari 2016, Murray and Rice 1993). But the framework's curvature obstruction $\hat{\kappa}$ for sensor fusion is *not* the derived Fisher–Ricci scalar of this geometry. It is a separate object: a dimensionless non-negative monotone function of $\mathrm{tr}(\mathbf{P})$, chosen for engineering convenience and motivated by the Fisher-geometric intuition, but defined as a design choice rather than derived as a curvature.

This distinction matters because earlier drafts of the framework conflated the two, and a careful review (documented in Appendix B and referenced in Chapter 20 §23.10) corrected the framing. We state the corrected position explicitly in this chapter and avoid claims the math does not support.

The chapter follows the standard sequence: the linear Kalman filter, its covariance Riccati equation, the Fisher-geometric reading and what it does and does not buy, the framework's specific $\hat{\kappa}$ design choice, and the ICARUS sensor-fusion sidebar reproduced verbatim. The chapter closes with a forward pointer to Chapter 12 on adaptive control.

§ 10.1

## *The linear Kalman filter*

Fix a discrete-time linear state-space system on $\mathbb{R}^n$ with state $\mathbf{x}_k \in \mathbb{R}^n$, control $\mathbf{u}_k \in \mathbb{R}^m$, and measurement $\mathbf{z}_k \in \mathbb{R}^p$:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{F}_k\,\mathbf{x}_k + \mathbf{G}_k\,\mathbf{u}_k + \mathbf{w}_k, & \mathbf{w}_k &\sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k), \\ \mathbf{z}_k &= \mathbf{H}_k\,\mathbf{x}_k + \mathbf{v}_k, & \mathbf{v}_k &\sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k). \end{aligned}$$

The matrices $\mathbf{F}_k, \mathbf{G}_k, \mathbf{H}_k, \mathbf{Q}_k, \mathbf{R}_k$ are known (possibly time-varying). The process noise $\mathbf{w}_k$ and measurement noise $\mathbf{v}_k$ are independent zero-mean Gaussian, with covariances $\mathbf{Q}_k \succeq 0$ and $\mathbf{R}_k \succ 0$. The initial state $\mathbf{x}_0 \sim \mathcal{N}(\hat{\mathbf{x}}_0^-, \mathbf{P}_0^-)$ is Gaussian and independent of all the noise.

**Discrete-time vs continuous-time.** The earlier chapters of this book formulate plants in continuous time ($\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$). Real flight controllers run discrete-time at fixed sample rates (typically 100 Hz–1 kHz), so the practical Kalman filter and controller live in the discrete formulation. The conversion is standard and not the framework's contribution: for a sample period $\Delta t$, the zero-order-hold discretization gives $\mathbf{F}_k = e^{\mathbf{A}\Delta t}$ and $\mathbf{G}_k = \int_0^{\Delta t} e^{\mathbf{A}\tau}\,d\tau\,\mathbf{B}$; the Tustin (bilinear) transform is the alternative when sample rates are slow relative to the plant's bandwidth. The continuous-time $\mathbf{Q}, \mathbf{R}$ become $\mathbf{Q}_k \approx \mathbf{Q}\,\Delta t$ and $\mathbf{R}_k \approx \mathbf{R}/\Delta t$ to first order. Every claim in this book transfers from continuous to discrete via these conversions; the framework's $\hat{\kappa}$ and modulation $m(C)$ are defined identically in both settings. The Apollo guidance computer ran a discrete-time Kalman at 25 Hz; modern avionics run it at 400–1000 Hz; the math is the same up to the discretization step.

The Kalman filter computes the conditional mean and covari-

ance of $\mathbf{x}_k$ given the measurement history $\mathbf{z}_1, \ldots, \mathbf{z}_k$, via two alternating steps.

**Time update (predict).** Given the posterior $(\hat{\mathbf{x}}^+_{k-1}, \mathbf{P}^+_{k-1})$ at the previous step, propagate forward through the dynamics:

$$\begin{aligned}\hat{\mathbf{x}}^-_k &= \mathbf{F}_{k-1}\,\hat{\mathbf{x}}^+_{k-1} + \mathbf{G}_{k-1}\,\mathbf{u}_{k-1},\\ \mathbf{P}^-_k &= \mathbf{F}_{k-1}\,\mathbf{P}^+_{k-1}\,\mathbf{F}^{\mathrm{T}}_{k-1} + \mathbf{Q}_{k-1}.\end{aligned}$$

The covariance grows by the process noise.

**Measurement update (correct).** Given the prior $(\hat{\mathbf{x}}^-_k, \mathbf{P}^-_k)$ and the new measurement $\mathbf{z}_k$, compute the innovation $\boldsymbol{\nu}_k = \mathbf{z}_k - \mathbf{H}_k\,\hat{\mathbf{x}}^-_k$, the innovation covariance $\mathbf{S}_k = \mathbf{H}_k\,\mathbf{P}^-_k\,\mathbf{H}^{\mathrm{T}}_k + \mathbf{R}_k$, and the Kalman gain

$$\mathbf{K}_k \;=\; \mathbf{P}^-_k\,\mathbf{H}^{\mathrm{T}}_k\,\mathbf{S}^{-1}_k.$$

The posterior is then

$$\begin{aligned}\hat{\mathbf{x}}^+_k &= \hat{\mathbf{x}}^-_k + \mathbf{K}_k\,\boldsymbol{\nu}_k,\\ \mathbf{P}^+_k &= (\mathbf{I} - \mathbf{K}_k\,\mathbf{H}_k)\,\mathbf{P}^-_k.\end{aligned}$$

The covariance shrinks by the information the measurement contributed.

The recursion is the entire filter. At every step the predicted mean and covariance are propagated forward; at every measurement the posterior is updated. The closed-loop form is exact for Gaussian noise and linear dynamics. For nonlinear systems, the extended Kalman filter linearizes the dynamics and measurements at each step; this is an approximation, but the algebraic structure is the same.

§ 10.2

## *The covariance Riccati equation*

The covariance propagation equations of §10.1 combine into a single recursion on $\mathbf{P}_k$ alone, with the state estimate $\hat{\mathbf{x}}_k$ falling out as a consequence. Eliminating intermediate quantities:

$$\mathbf{P}_k^+ \;=\; \mathbf{P}_k^- - \mathbf{P}_k^- \,\mathbf{H}_k^{\mathrm{T}} \big(\mathbf{H}_k \,\mathbf{P}_k^- \,\mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k\big)^{-1} \mathbf{H}_k \,\mathbf{P}_k^-.$$

This is the *discrete Riccati update*. In steady state (time-invariant $\mathbf{F}, \mathbf{H}, \mathbf{Q}, \mathbf{R}$), the recursion converges to a fixed-point covariance $\mathbf{P}_\infty$ that satisfies the discrete algebraic Riccati equation:

$$\mathbf{P}_\infty \;=\; \mathbf{F}(\mathbf{P}_\infty - \mathbf{P}_\infty \mathbf{H}^{\mathrm{T}} (\mathbf{H}\mathbf{P}_\infty \mathbf{H}^{\mathrm{T}} + \mathbf{R})^{-1} \mathbf{H}\mathbf{P}_\infty)\mathbf{F}^{\mathrm{T}} + \mathbf{Q}.$$

Compare this to LQR's CARE from Chapter 8 §8.2. The structural similarity is exact: the Kalman covariance Riccati is the LQR Riccati under the substitution

$$\mathbf{A} \leftrightarrow \mathbf{F}^{\mathrm{T}}, \quad \mathbf{B} \leftrightarrow \mathbf{H}^{\mathrm{T}}, \quad \mathbf{Q}_{\mathrm{LQR}} \leftrightarrow \mathbf{Q}_{\mathrm{Kalman}}, \quad \mathbf{R}_{\mathrm{LQR}} \leftrightarrow \mathbf{R}_{\mathrm{Kalman}}.$$

This is the famous *LQR/Kalman duality* (Kalman 1960, formalized later). The control problem of driving a state to zero, and the estimation problem of estimating a state from noisy measurements, are dual: solving one solves the other under the substitution above. The same algebraic-Riccati machinery, the same Newton–Kleinman iteration (Chapter 8 §8.3), the same gauge-dependence/gauge-invariance distinction.

**The continuous-time version.** For continuous-time systems $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{w}$, $\mathbf{z} = \mathbf{H}\mathbf{x} + \mathbf{v}$, the covariance evolves according to a differential Riccati equation:

$$\dot{\mathbf{P}}(t) \;=\; \mathbf{A}\mathbf{P} + \mathbf{P}\mathbf{A}^{\mathrm{T}} - \mathbf{P}\mathbf{H}^{\mathrm{T}}\mathbf{R}^{-1}\mathbf{H}\mathbf{P} + \mathbf{Q}.$$

At steady state, $\dot{\mathbf{P}} = 0$ gives the continuous algebraic Riccati equation, which is the exact dual of the LQR CARE from Chapter 8.

§ 10.3

## *The Fisher manifold: what the information geometry buys*

The posterior at each time is a Gaussian distribution $\mathcal{N}(\hat{\mathbf{x}}, \mathbf{P})$ over the state space, parameterized by its mean $\hat{\mathbf{x}}$ and covariance $\mathbf{P}$. The set of all such Gaussian distributions forms a finite-dimensional manifold called the *Gaussian parameter manifold* or, in the literature of information geometry, the *Fisher manifold* (Amari 2016; Murray and Rice 1993).

**The Fisher information matrix.** For a parametric family $p(\mathbf{x}|\boldsymbol{\theta})$ with parameter $\boldsymbol{\theta} \in \mathbb{R}^d$, the Fisher information matrix is

$$\mathbf{I}(\boldsymbol{\theta}) \;=\; \mathbb{E}\Big[\, \big(\nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\boldsymbol{\theta})\big)\big(\nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\boldsymbol{\theta})\big)^{\mathrm{T}} \,\Big].$$

For a Gaussian with known covariance and unknown mean, the Fisher information is $\mathbf{I}(\hat{\mathbf{x}}) = \mathbf{P}^{-1}$. The covariance and the inverse Fisher information are the same object in the linear-Gaussian limit. The Cramér–Rao bound says that any unbiased estimator's covariance is bounded below (in the Loewner order) by $\mathbf{I}^{-1} = \mathbf{P}$; the Kalman filter achieves this bound.

**The Fisher metric and curvature.** The Fisher information matrix defines a Riemannian metric on the parameter manifold:

$$g_{\mathbf{I}}(\delta\boldsymbol{\theta}_1, \delta\boldsymbol{\theta}_2) \;=\; \delta\boldsymbol{\theta}_1^{\mathrm{T}}\, \mathbf{I}(\boldsymbol{\theta})\, \delta\boldsymbol{\theta}_2,$$

under which the parameter manifold becomes a Riemannian space. This space has curvature: the Levi-Civita connection from the Fisher metric has a Riemann curvature tensor, which contracts to a Ricci tensor, which contracts to a scalar—the *Fisher–Ricci scalar* $R_{\text{Fisher}}$. For the Gaussian family on $\mathbb{R}^n$ parameterized by $(\hat{\mathbf{x}}, \mathbf{P})$, the Fisher–Ricci scalar can be computed in closed form[2] and is generically nonzero.

**What this geometry buys.** Several useful structures fall out:

- The Kalman update is, in the Fisher metric, an approximate natural-gradient step toward the maximum-likelihood mean given the observed measurement. The natural-gradient interpretation is exact in the linear-Gaussian limit and approximate (but well-motivated) for nonlinear extensions.

- The trace of the covariance, $\text{tr}(\mathbf{P})$, is a scalar invariant of the posterior distribution that measures total uncertainty. It is gauge-dependent in the sense of Chapter 6 (rescaling state coordinates rescales $\text{tr}(\mathbf{P})$), but its monotonicity properties—grows with process noise, shrinks with informative measurements—are gauge-invariant.

- The Cramér–Rao bound's saturation by Kalman gives a geometric statement: *the Kalman filter is the natural-gradient estimator on the Fisher manifold in the linear-Gaussian limit.* This is the cleanest information-geometric reading of the filter.

[2]The exact formula depends on convention. For the mean-only submanifold (covariance held fixed), the manifold is flat; the Fisher–Ricci scalar is identically zero. For the covariance manifold (mean held fixed), the manifold has constant negative scalar curvature proportional to $-n(n+1)/2$ on the $n \times n$ symmetric positive-definite cone (Skovgaard 1984). For the full $(\hat{\mathbf{x}}, \mathbf{P})$ manifold, the scalar is a function of both that depends on the parameterization. The exact value is convention-dependent; the sign is convention-invariant.

**What this geometry does *not* directly buy.** The Fisher–Ricci scalar $R_{\text{Fisher}}$ is a real curvature with a real geometric interpretation, but it is signed, convention-dependent, and not naturally dimensionless. Earlier drafts of the framework attempted to define the sensor-fusion curvature obstruction $\hat{\kappa}$ as directly proportional to $R_{\text{Fisher}}$ (or to $-R_{\text{Fisher}}$, depending on convention). This was an error, identified during a careful review and corrected in the v3 framework documentation[3]. The next section gives the corrected design choice the framework actually uses.

§ 10.4

## *The framework's $\hat{\kappa}$ for sensor fusion: a design choice*

The Davis-Geometric framework's curvature obstruction $\hat{\kappa}$ for the sensor-fusion bundle is defined as

$$\hat{\kappa}(\mathbf{P}) \;=\; f\big(\operatorname{tr}(\mathbf{P})\big),$$

where $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a monotonically non-decreasing function chosen by the designer. In the simplest case, $f$ is the identity scaled by a baseline and clamped to be non-negative:

$$\hat{\kappa}(\mathbf{P}) \;=\; \max\!\left(0,\; \frac{\operatorname{tr}(\mathbf{P})}{\operatorname{tr}(\mathbf{P}_{\text{baseline}})} - 1\right),$$

[3]The correction is documented in Chapter 20 §23.10, item (2). The error was: defining $\hat{\kappa}$ as a literal Fisher–Ricci scalar conflates a signed convention-dependent quantity with a non-negative dimensionless design parameter. The corrected framing makes $\hat{\kappa}$ a design choice motivated by Fisher geometry but defined separately. See Appendix B for the full corrected framing.

where $\mathbf{P}_{\text{baseline}}$ is the covariance at nominal operating conditions. The clamp matters: without it, periods when the filter is doing *better* than baseline would produce $\hat{\kappa} < 0$ and $C = 1/(1+\hat{\kappa}) > 1$, contradicting the framework's requirement that confidence lie in $(0, 1]$. With the clamp, $\hat{\kappa} = 0$ when the filter is operating at or better than baseline (full confidence), $\hat{\kappa}$ is positive when the covariance has grown beyond nominal, and unbounded as the filter loses track.

The operational confidence $C = 1/(1+\hat{\kappa})$ is then 1 at baseline, drops toward 0 as $\text{tr}(\mathbf{P})$ grows, and gates the controller's gain by $m(C) \cdot \mathbf{K}_{\text{lqr}}$ exactly as Chapter 8 §8.6 described.

**Why this design choice and not the Fisher–Ricci scalar.** Three reasons:

1. *Sign.* $\hat{\kappa}$ is required to be non-negative (a confidence-degrading obstruction). The Fisher–Ricci scalar is signed; in some conventions it is negative for the Gaussian parameter manifold's covariance sub-component. Using a signed quantity directly would require ad-hoc sign-fixing.

2. *Dimensionlessness.* $\hat{\kappa}$ is dimensionless by design. $\text{tr}(\mathbf{P})$ has units of $(\text{state})^2$; dividing by $\text{tr}(\mathbf{P}_{\text{baseline}})$ produces a dimensionless ratio. (Caveat: $\text{tr}(\mathbf{P})$ is not a coordinate-invariant geometric quantity—it depends on the chosen state coordinates and their relative scaling. The framework treats the baseline-normalized ratio as a fixed-coordinate operational scalar, not as a Riemannian invariant. A coordinate-invariant alternative is the whitened trace $\text{tr}(\mathbf{P}_{\text{baseline}}^{-1}\mathbf{P})$, which is invariant under linear changes of coordinates and is what an implementation should use when the state coordinates are not pre-whitened.) The Fisher–Ricci scalar has dimension that depends on the parameterization and would require a separate nondimensionalization.

3. *Engineering observability.* $\mathrm{tr}(\mathbf{P})$ is directly available from any Kalman filter implementation; the Fisher–Ricci scalar requires the full Riemann tensor computation, which is expensive and parameterization-dependent. The framework optimizes for "computable in tens of microseconds in the flight loop," which means using the directly available scalar rather than the derived one.

**What the design choice loses.** It does lose the precise curvature interpretation. The framework cannot claim $\hat{\kappa}$ *is* a curvature in the differential-geometric sense; it can only claim $\hat{\kappa}$ is *motivated by* curvature geometry. The design choice is what the engineering substrate uses; the curvature interpretation is what the geometric reading offers as intuition. The two are kept separate to avoid conflating them.

**What the design choice keeps.** The structural role of $\hat{\kappa}$ as the substrate's uncertainty scalar is preserved. The operational confidence $C = 1/(1+\hat{\kappa})$ still gates the controller. The Curvature Floor inequality of Chapter 7 §7.5 still applies; $\hat{\kappa}$ in the inequality is exactly the framework's design-choice $\hat{\kappa}$, not a derived Fisher–Ricci scalar. The chapter's reading remains operational: large $\mathrm{tr}(\mathbf{P})$ means the filter is uncertain about the state, the controller should reduce its aggressiveness, the modulation $m(C) < m_{\max}$ does so.

§ 10.5

## *Sidebar: ICARUS sensor-fusion implementation*

*The following sidebar is reproduced from the ICARUS architecture documentation.*

Sensor fusion residuals are pre-computed by the sensor thread and stored as fiber fields on the `sensor_fusion` bundle. The flight loop reads them via `SECTION sensor_fusion AT t = @latest_t`. The bundle's local curvature obstruction $\hat{\kappa}$—derived from the `CURVATURE sensor_fusion` query—provides a dimensionless non-negative uncertainty scalar that, in the linear-Gaussian limit, can be defined as a monotone function of $\mathrm{tr}(\mathbf{P})$ (a *design choice*, not a derived Fisher Ricci scalar—see §23.10(2) for the corrected framing). Appendix B states the corrected Fisher-geometry facts and the design choice explicitly.

**Architectural unpacking.** ICARUS separates the sensor thread (typically running at 200–1000 Hz, depending on sensor bandwidth) from the flight loop (typically 200–400 Hz). The sensor thread runs the Kalman update against incoming IMU, GNSS, and air-data measurements; the resulting posterior covariance $\mathbf{P}$ and the innovation residuals are pushed to a `sensor_fusion` bundle in GIGI. The flight loop queries this bundle through GQL: `SECTION sensor_fusion AT t = @latest_t` returns the most recent posterior. `CURVATURE sensor_fusion` returns $\hat{\kappa}$, computed by the substrate as the design-choice function of $\mathrm{tr}(\mathbf{P})$.

This architecture means the flight loop never solves a Kalman update at flight time. The heavy estimation work is done in the sensor thread; the flight loop's interaction with the filter is a single section-read and a single curvature-read per cycle. Both are constant-time GIGI queries.

The verbatim sidebar disclosure—"a *design choice*, not a derived Fisher Ricci scalar"—is the framework's explicit honesty about the structure-vs-derivation distinction of §10.4. The substrate is doing the engineering; the geometry is doing the

motivation. The two are not conflated.

§ 10.6

## *Forward: estimation, adaptation, and the matching condition*

The Kalman filter is for *estimation*: given a known plant model, estimate the state from noisy measurements. The next set of chapters (the remainder of Part IV) introduces *adaptation*: given an *uncertain* plant model, simultaneously estimate the state and the model parameters, and adjust the controller as the estimate improves.

The bridge from estimation to adaptation is the *matching condition*: a structural assumption that the uncertain parameters in the plant model couple to the input channel in a specific way that lets adaptive feedback compensate for them without destabilizing the closed-loop. Chapter 11 develops the matching condition formally. Chapter 12 applies it to Model-Reference Adaptive Control (MRAC), which is the standard textbook adaptive law. Chapter 13 returns to the Curvature Floor inequality with the adaptive terms ($\alpha \operatorname{tr}(\mathbf{\Gamma}^{-1})$ and $\beta \operatorname{curv}(\bar{\Theta})$) reinstated, completing the inequality form left as a forward pointer in Chapter 7 §7.5.

The Fisher-geometric reading of this chapter generalizes: where Kalman has a fixed parameter manifold (the Gaussian family, parameterized by mean and covariance), adaptive control has a parameter manifold that the adaptive law is searching across. The framework's curvature obstruction $\hat{\kappa}$ for adaptive control will involve curvature of the parameter manifold $\bar{\Theta}$—and that curvature, unlike the Fisher–Ricci scalar of §10.3, *is* part of the inequality. The Curvature Floor's $\beta \operatorname{curv}(\bar{\Theta})$ term is exactly

the curvature of the adaptive parameter manifold under the Fisher metric induced by the matching condition. The framework's distinction between design-choice $\hat{\kappa}$ (for sensor fusion) and curvature-as-such (for adaptation) is preserved.

## § 10.7

## *What you take from this chapter*

Three things.

**First.** The Kalman filter is the closed-form recursive Bayesian update for linear-Gaussian systems. The covariance $\mathbf{P}_k$ satisfies a discrete Riccati equation that, in steady state, is the algebraic dual of the LQR CARE under the substitution $\mathbf{A} \leftrightarrow \mathbf{F}^{\mathrm{T}}$, $\mathbf{B} \leftrightarrow \mathbf{H}^{\mathrm{T}}$. The same Newton–Kleinman machinery solves both.

**Second.** The Gaussian parameter manifold has a Fisher metric, under which the covariance $\mathbf{P}$ is the inverse Fisher information and the Kalman update is an approximate natural-gradient step. The manifold has a Fisher–Ricci scalar curvature that is signed and convention-dependent. The framework's curvature obstruction $\hat{\kappa}$ for sensor fusion is *not* the derived Fisher–Ricci scalar; it is a design choice defined as a monotone function of $\mathrm{tr}(\mathbf{P})$, chosen to be non-negative, dimensionless, and directly observable from the filter's state. The distinction is documented in Appendix B and acknowledged in the ICARUS architecture's verbatim sidebar.

**Third.** ICARUS's sensor-fusion architecture stores the filter output on a GIGI bundle. The flight loop reads sections of this bundle via GQL queries and reads the curvature obstruction $\hat{\kappa}$ as a single substrate-computed scalar. The Kalman computation

lives in the sensor thread; the flight loop never solves a Riccati at flight time. This is the same offline-versus-runtime architectural commitment Chapter 8 §8.6 named for the LQR case, applied to the dual estimation problem.

The next chapter develops the matching condition—the bridge from estimation to adaptation—and prepares the way for MRAC, the Curvature Floor with adaptive terms, and the rest of Part IV.

*Vamos.*

Chapter 11

# *Lyapunov methods and the matching condition*

*Lyapunov's method allows us to analyze the stability of equilibrium points without solving the differential equation.*
*Hassan K. Khalil, Nonlinear Systems (3rd edition, 2002), Chapter 4*

## § 11.0

## *Stability without integration*

The kid in §1.0 of this book did Lyapunov's method without knowing what it was called.

Remember the lawnmower-on-BMX scene. The kid had bolted a lawnmower engine onto a bicycle frame. She sat on it, considered pulling the starter, and imagined what would happen: the engine would catch, the bike would accelerate, she had no brakes, the wall at the end of the driveway was thirty feet away, her mother would find her. The kid concluded—without ever pulling the starter, without ever rolling forward an inch—that

the configuration was unstable. She chose not to ride.

Look at what the kid did, mechanically. She did NOT solve the bike's equation of motion. She did not compute the acceleration, integrate the velocity, integrate the position, find the time-of-arrival at the wall. She did none of that. What she did was identify a property of the system—the kinetic energy that would build up under the engine's thrust—and reason about its growth against the geometry of the driveway. She imagined an energy-like quantity and reasoned about its behavior along whatever trajectory the system was going to take, without ever computing the trajectory itself.

That is Lyapunov's method, in its entirety.

**A second example, the one most textbooks open with.** You are sitting at the top of a hill on a bicycle. The brakes are good. You let them go. You do not need to solve any differential equation to know roughly what will happen:

- The bike has potential energy (your height above the bottom of the hill).
- As you roll, that potential energy converts into kinetic energy (speed).
- Gravity pulls you downhill; friction and air drag slow you.
- You end up somewhere near the bottom of the hill, going some speed, eventually slowing to a stop in the flat.

You knew all of this without ever writing down the equation of motion. You knew it because there is a quantity—the total energy (potential plus kinetic)—that you can reason about without solving for the trajectory. The total energy decreases over time (because friction dissipates it). It is bounded below by zero. So the energy converges to some final value, and the bike's state converges to whatever configuration corresponds to that final

energy. You have just concluded the bike's long-term behavior without ever integrating its dynamics.

This is the structure of every Lyapunov argument: *find a scalar quantity that decreases along trajectories, conclude where the trajectory ends up, without ever solving for the trajectory.*

**Why this is a mathematical revolution.** For most nonlinear systems—the kind of systems flight control actually faces—the equation of motion has no closed-form solution. You can't write down $x(t)$. Standard analytical machinery, the kind Newton invented, does not work. If you want to know whether the system stays bounded, you would have to either simulate it (which only tests specific initial conditions) or give up.

Lyapunov's idea changed this. You don't need to solve $x(t)$. You need to find a function $V(x)$ on the state space—a scalar valued "energy"—such that:

- $V$ is positive everywhere except at the equilibrium (positive-definite).
- $V$ decreases along any trajectory the system can produce (its time-derivative is non-positive).

If you can find such a $V$, the system is stable: trajectories stay bounded, and in many cases converge to the equilibrium. You did not have to solve a single ODE. You replaced one impossible problem (solve $x(t)$ in closed form) with a different problem (find a $V$ that decreases). The new problem is often solvable when the old one is not.

The trade is: you give up knowing where the trajectory *is*, in exchange for knowing where it *can't go*. The level sets of $V$ are nested closed surfaces around the equilibrium; once a trajectory is inside one, it can never leave (because $V$ would have to increase to leave, and we have prevented that). The kid, looking at the lawnmower-bike, saw the level set she was about to enter, looked

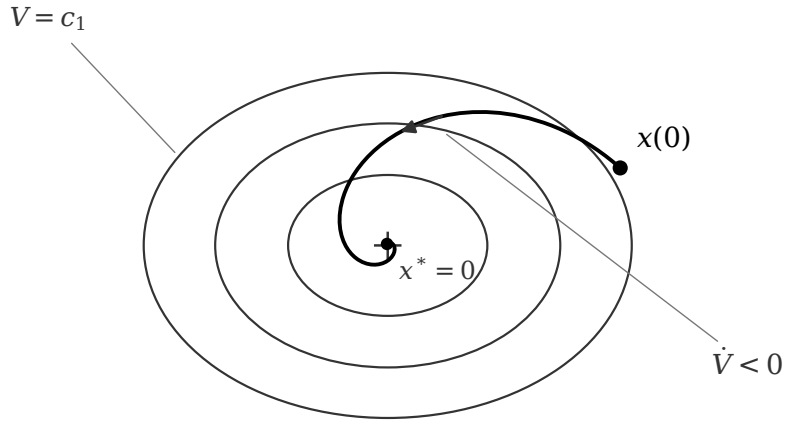



Figure 11.1: A Lyapunov function and a descending trajectory. The nested ellipses are level sets $V(\mathbf{x}) = c$ of a quadratic Lyapunov function. The trajectory spirals inward, crossing each level set in the direction of decreasing $V$ ($\dot{V} < 0$). The picture says: find a quantity that decreases along trajectories, and conclude convergence to the equilibrium without ever solving the differential equation.

at the wall, and decided not to start the engine. The same move, made formal, is what we will write down in §11.1.

**The history.** Aleksandr Mikhailovich Lyapunov defended his doctoral thesis at Kharkov University in 1892. The thesis, titled *The General Problem of the Stability of Motion*, introduced this method. It was almost entirely ignored for half a century. Lyapunov died in 1918; the thesis was reprinted, ignored, and finally noticed in the 1950s by the Russian school of control theory—Krasovskii, Pontryagin, Letov—who realized it was exactly the tool the emerging field of nonlinear control was missing. By the 1970s it had become the standard analytical machinery for nonlinear stability proofs; by the 1990s it was the foundation of every adaptive-control textbook.

The reason for the long delay is the reason the method is interesting. Lyapunov's idea is structurally unlike most analytical mathematics. It is closer in spirit to thermodynamics than to differential equations. Newton would have looked at it sideways. Hamilton would have understood it. The twentieth century needed it.

This chapter develops Lyapunov methods in the form they will be used in the rest of Part IV. The exposition is compact—there are many excellent introductions (Khalil, Sastry, Vidyasagar) and I will not redo what they do well. What I will do is set up the specific tools needed for the chapters that follow: the Lyapunov stability theorem in its useful form, the ultimate-boundedness extension (which is the right tool for adaptive control where the state stays bounded but does not converge to zero), the composite Lyapunov function on a coupled state-plus-parameter manifold, and the structural assumption called the *matching condition* that makes adaptive feedback possible. The chapter closes with the limitations of the method: what Lyapunov tells you, and what it does not.

§ 11.1

## *The Lyapunov stability theorem*

Consider an autonomous nonlinear system on $\mathbb{R}^n$:

$$\dot{\mathbf{x}}(t) \;=\; \mathbf{f}(\mathbf{x}(t)), \qquad \mathbf{f}(\mathbf{0}) = \mathbf{0}.$$

The origin is an equilibrium. We want to know whether trajectories starting near the origin stay near the origin (stability), or converge to it (asymptotic stability).

**Definition (Lyapunov function).** A scalar-valued function $V : \mathbb{R}^n \to \mathbb{R}$ is a *Lyapunov function* for the system at the origin if:

1. $V$ is continuously differentiable on a neighborhood $\mathcal{D}$ of the origin.
2. $V(\mathbf{0}) = 0$ and $V(\mathbf{x}) > 0$ for all $\mathbf{x} \in \mathcal{D} \setminus \{\mathbf{0}\}$ (positive definite).
3. $\dot{V}(\mathbf{x}) = \nabla V(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x}) \le 0$ for all $\mathbf{x} \in \mathcal{D}$ (negative semi-definite along trajectories).

**Theorem (Lyapunov, 1892).** If a Lyapunov function exists for the system at the origin, then the origin is *stable in the sense of Lyapunov*: for every $\epsilon > 0$ there is $\delta > 0$ such that $\|\mathbf{x}(0)\| < \delta$ implies $\|\mathbf{x}(t)\| < \epsilon$ for all $t \ge 0$. If in addition $\dot{V}(\mathbf{x}) < 0$ on $\mathcal{D} \setminus \{\mathbf{0}\}$, the origin is *asymptotically stable*: trajectories starting near the origin converge to it.

The proof is short and worth knowing. Pick any $\epsilon > 0$. Choose $\eta < \min_{\|\mathbf{x}\| = \epsilon} V(\mathbf{x})$, which is positive by the positive-definiteness of $V$. Then the sublevel set $\{V \le \eta\}$ is contained in the open ball of radius $\epsilon$. Pick $\delta$ small enough that $\{\|\mathbf{x}\| < \delta\} \subseteq \{V \le \eta\}$.

Then any trajectory starting in the $\delta$-ball stays in $\{V \leq \eta\}$ forever (because $\dot{V} \leq 0$ means $V$ is non-increasing along the trajectory), and therefore stays in the $\epsilon$-ball. Asymptotic stability follows from a slightly tighter argument with $\dot{V} < 0$ off the equilibrium.

What is happening structurally: the level sets $\{V = c\}$ are nested closed surfaces around the origin. The condition $\dot{V} \leq 0$ says the trajectory cannot cross any level set outward. Once inside a level set, you stay there. The level sets foliate the neighborhood of the origin into nested invariant regions; the trajectory's level can only decrease.

**What this buys us.** We have concluded stability of the equilibrium without ever computing a trajectory of $\mathbf{f}$. The Lyapunov function $V$ is the artifact; the dynamical behavior is read off its properties. This is exactly what makes the method useful for control: in a closed-loop system, we get to design the controller, and we can sometimes design it to make a known $V$ satisfy the stability conditions.

This is the strategy of Lyapunov-based control design: pick a candidate $V$, choose the controller to enforce $\dot{V} \leq 0$, and conclude stability. The design problem is reduced to algebra on $V$ and the closed-loop dynamics.

## § 11.2

## *Ultimate boundedness*

Go back to the hill from §11.0 for a moment. Imagine that instead of a smooth slope down to flat ground, the hill has small bumps and ruts in it—random little disturbances that nudge the bike sideways as it rolls. The bike still rolls downhill, but it doesn't end up at a single point at the bottom; it wobbles

around in a small region near the bottom, deflected this way and that by the bumps.

That is *ultimate boundedness*. The bike doesn't converge to a single resting point—the bumps don't let it—but the bike does end up confined to a bounded region, and once it's in that region, it never leaves. The size of the region depends on how big the bumps are.

This is the form of stability adaptive control actually achieves. Parameter estimates wander; tracking error stays small but does not vanish. We don't get convergence to a single point. We get a bounded region the state ends up in, with the size of the region set by the disturbance budget. The formal statement:

**Definition.** The system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ is *ultimately uniformly bounded* with ultimate bound $b > 0$ if for every $a > 0$ there exists $T = T(a) \geq 0$ such that

$$\|\mathbf{x}(0)\| \leq a \implies \|\mathbf{x}(t)\| \leq b \quad \text{for all } t \geq T(a).$$

**Theorem (Lyapunov ultimate boundedness).** Suppose there exists a continuously differentiable $V : \mathbb{R}^n \to \mathbb{R}$ and positive constants $\alpha_1, \alpha_2, \alpha_3, \mu$ such that:

$$\begin{aligned} \alpha_1\|\mathbf{x}\|^2 \;\leq\; V(\mathbf{x}) \;&\leq\; \alpha_2\|\mathbf{x}\|^2, \\ \dot{V}(\mathbf{x}) \;&\leq\; -\alpha_3\|\mathbf{x}\|^2 + \eta, \qquad \text{for some constant } \eta > 0. \end{aligned}$$

Then the system is ultimately uniformly bounded with ultimate bound $b = \sqrt{\alpha_2\eta/(\alpha_1\alpha_3)}$.

The proof structure is the same as for stability, with the modification that $\dot{V}$ is not negative everywhere—it is negative only outside a ball around the origin (the region where $\alpha_3\|\mathbf{x}\|^2 > \eta$). Inside that ball, $V$ may grow, but only up to the limit set by the constants. The trajectory cannot escape the corresponding $V$-sublevel set.

**Why this matters operationally.** The Curvature Floor inequality of Chapter 7 §7.5 has exactly this form for adaptive systems. The quadratic boundary $\alpha_3\|\mathbf{x}\|^2 = \eta$ is the operational closed-loop envelope: outside it, the Lyapunov function decreases (the controller is working); inside it, the Lyapunov function may grow modestly (the controller is fighting the disturbance, not winning, not losing). The ultimate-bound $b$ is the framework's operational version of "how big can the tracking error get, in the worst case, given the disturbance budget $\eta$."

The quadratic boundary equation $\alpha \cdot m_{\min} \cdot r = \beta \cdot r^2 + \eta \cdot D_{\max}$ that appears in the ICARUS performance documentation is exactly this kind of Lyapunov ultimate-boundedness statement, specialized to the curvature-modulated gain $m_{\min}$ and the disturbance bound $D_{\max}$. The framework reads it as a curvature-aware bound on the closed-loop envelope size.

§ 11.3

## *The composite Lyapunov function*

Adaptive control couples two state spaces: the plant state $\mathbf{x}$ and the parameter estimate $\hat{\boldsymbol{\theta}}$ that the adaptive law is trying to refine. The joint state lives on $\mathbb{R}^n \times \mathbb{R}^d$, where $d$ is the dimension of the unknown parameter vector $\boldsymbol{\theta}$.

A Lyapunov function for the joint system must be positive-definite on the joint state. The standard construction is to combine a quadratic in $\mathbf{x}$ with a quadratic in the parameter error $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} - \hat{\boldsymbol{\theta}}$:

$$V(\mathbf{x}, \tilde{\boldsymbol{\theta}}) \;=\; \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x} \;+\; \tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\tilde{\boldsymbol{\theta}},$$

where $\mathbf{P} \succ 0$ is the LQR-like state-weighting matrix and $\boldsymbol{\Gamma} \succ 0$ is the adaptive law's gain matrix. This is the *composite Lya-*

*punov function.* It is positive-definite on the joint state because both summands are positive-definite in their arguments and add directly.

**The structural insight.** The composite function says: stability of the adaptive system is the simultaneous statement that the plant state's quadratic Lyapunov function $\mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ stays bounded *and* the parameter error's quadratic Lyapunov function $\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\tilde{\boldsymbol{\theta}}$ stays bounded. Each term tracks one of the two coupled subsystems; the gradient of $V$ with respect to each contributes to $\dot{V}$.

The adaptive law is chosen so that the cross terms in $\dot{V}$ cancel, leaving only a negative-definite contribution from the plant state. Specifically: if the parameter error $\tilde{\boldsymbol{\theta}}$ enters the plant dynamics linearly, and the adaptive law sets $\dot{\hat{\boldsymbol{\theta}}}$ proportional to a state-dependent regressor, then the cross terms in $\dot{V}$ vanish identically, and $\dot{V} = -\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} \leq 0$, where $\mathbf{Q}$ is the algebraic-Lyapunov-equation matrix associated with $\mathbf{P}$.

**What the composite construction provides.** Boundedness of $V$ along trajectories implies boundedness of both $\mathbf{x}$ and $\tilde{\boldsymbol{\theta}}$. Asymptotic decrease of $V$ (when $\mathbf{x} \neq \mathbf{0}$) implies $\mathbf{x} \to \mathbf{0}$ via Barbalat's lemma. The parameter error $\tilde{\boldsymbol{\theta}}$ stays bounded but does not necessarily converge to zero—that requires the additional condition of persistency of excitation, which Chapter 12 develops.

This is the central technical structure of adaptive control. The composite Lyapunov function is the geometric content; everything else is computing on it.

§ 11.4

## *The matching condition*

For the composite Lyapunov construction to produce a clean $\dot{V}$, the parameter uncertainty must enter the plant dynamics in a specific structural way. This is the *matching condition*.

**The bike-and-basket picture.** Imagine you are riding a bike, and someone has slipped an unknown weight into your basket. You don't know how heavy it is. Two scenarios:

- *The weight just sits in the basket*. It makes the bike harder to accelerate—you have to pedal harder to maintain speed. But it doesn't make the bike steer differently; it doesn't make the wheels wobble. The mystery weight affects you *through the same channel* that your pedaling does (the propulsion direction). So if you knew the weight, you could just pedal a bit harder to compensate. Even not knowing it, you can FEEL that the bike is slower than usual, infer there's extra mass, and pedal accordingly. This is the *matched* case.

- *The weight is lopsided and pulls the bike sideways*. Now the unknown is acting through a channel you can't reach with pedaling. No amount of harder pedaling fixes the sideways pull; you'd need a different actuator (lean, steer, hand on the basket) to compensate. This is the *unmatched* case.

The matching condition is the formal way of saying "the unknown enters through the same channel as the control." When matching holds, the controller can compensate for the unknown by adjusting its existing control inputs. When matching fails, the controller is blind to a piece of the dynamics, and direct adaptive feedback can't fix it.

For aerospace problems, matching usually holds for things like mass changes (fuel burn) and aerodynamic-coefficient drift—the uncertain physics enters through the same input channels (thrust, control-surface deflections) that the controller already commands. It often fails for things like asymmetric damage or sensor faults—those create new failure channels the original controller wasn't designed to reach.

The framework's claim is: when matching holds, use direct adaptive feedback (MRAC, Chapter 12); when matching fails, fall back on the curvature-modulated robust control of Chapter 8, which doesn't need matching. The substrate (ICARUS) lets you switch between them based on what the runtime curvature obstruction is telling you about the regime.

**Definition (formal).** Consider a plant with state $\mathbf{x} \in \mathbb{R}^n$, control $\mathbf{u} \in \mathbb{R}^m$, and unknown parameter vector $\boldsymbol{\theta} \in \mathbb{R}^d$:

$$\dot{\mathbf{x}} \;=\; \mathbf{A}\,\mathbf{x} \;+\; \mathbf{B}\big(\mathbf{u} \;+\; \mathbf{Y}(\mathbf{x})\,\boldsymbol{\theta}\big),$$

where $\mathbf{Y}(\mathbf{x}) \in \mathbb{R}^{m\times d}$ is a known matrix-valued regressor (the *regressor matrix*, sometimes written $\boldsymbol{\Phi}(\mathbf{x})$) and $\boldsymbol{\theta} \in \mathbb{R}^d$ is the unknown parameter vector. The product $\mathbf{Y}(\mathbf{x})\,\boldsymbol{\theta}$ takes values in $\mathbb{R}^m$, so it can be added to $\mathbf{u}$ inside the $\mathbf{B}$ input channel. In the scalar SISO case ($m = 1$, $\mathbf{Y}(\mathbf{x})$ a row vector), this reduces to the more common $\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$ form. The system is said to satisfy the *matching condition* because the uncertain term enters through the same input channel $\mathbf{B}$ as the control $\mathbf{u}$.

**Why "matching."** The control input can directly cancel the uncertain term. If the parameter estimate $\hat{\boldsymbol{\theta}}$ is correct, the control law

$$\mathbf{u} \;=\; -\mathbf{K}\mathbf{x} - \mathbf{Y}(\mathbf{x})\,\hat{\boldsymbol{\theta}}$$

collapses the plant dynamics to

$$\dot{\mathbf{x}} \;=\; (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} \;+\; \mathbf{B}\,\mathbf{Y}(\mathbf{x})(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \;=\; \mathbf{A}_m\mathbf{x} \;+\; \mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}},$$

where $\mathbf{A}_m = \mathbf{A} - \mathbf{BK}$ is the closed-loop nominal matrix (Hurwitz by the choice of $\mathbf{K}$) and $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} - \hat{\boldsymbol{\theta}}$ is the parameter error. The uncertainty has been collapsed to a single multiplicative term, multiplied by $\mathbf{B}$ on the left. The adaptive law's job is to drive $\tilde{\boldsymbol{\theta}}$ small.

**Why the condition is restrictive.** Matching is a strong structural assumption. It requires the uncertain dynamics to be commensurable with the input channel. Many real systems do not satisfy it. Examples:

- *Unmatched parametric uncertainty.* If the uncertain term enters through a different channel, e.g. $\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu} + \mathbf{B}_\theta \mathbf{Y}(\mathbf{x})\boldsymbol{\theta}$ with $\mathbf{B}_\theta \neq \mathbf{B}$, no direct cancellation is possible. Such systems are studied under the heading of *unmatched uncertainty* and require more elaborate adaptive laws.

- *Nonlinear matching failure.* If the uncertain term is $\mathbf{Y}(\mathbf{x}, \mathbf{u})\,\boldsymbol{\theta}$ depending on the control itself, the linear-in-$\boldsymbol{\theta}$ structure still holds for fixed $\mathbf{u}$ but the closed-loop algebra no longer collapses cleanly, and the composite Lyapunov construction does not produce the same clean $\dot{V}$ cancellation.

- *Time-varying parameter.* If $\boldsymbol{\theta}$ varies with time (the true parameter is not a constant the adaptive law is searching for), the matching framework requires modification (typically a $\sigma$-modification term or a deadzone).

**Why we use it anyway.** For many aerospace problems—trim-tracking with uncertain aerodynamic coefficients, mass changes from fuel consumption, control-effectiveness loss from damage—the parametric uncertainty does enter through the input channel to good approximation. The matching condition is restrictive but covers a useful design envelope. When it fails, the practical move is usually to either nondimensionalize the uncertainty

until it matches, or to use a gain-scheduled rather than fully adaptive controller, or to fall back on the curvature-modulated robust approach of the framework—which is exactly the ICARUS architectural commitment.

§ 11.5

## *The closed-loop stability proof for matched uncertainty*

Combining the pieces of §11.3 and §11.4: under the matching condition, with the nominal control law $\mathbf{u} = -\mathbf{K}\mathbf{x} - \mathbf{Y}(\mathbf{x})\,\hat{\boldsymbol{\theta}}$, the closed-loop dynamics are

$$\dot{\mathbf{x}} \;=\; \mathbf{A}_m\mathbf{x} \;+\; \mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}}.$$

Let $\mathbf{P} \succ 0$ solve the algebraic Lyapunov equation

$$\mathbf{A}_m^{\mathrm{T}}\mathbf{P} + \mathbf{P}\mathbf{A}_m \;=\; -\mathbf{Q}, \qquad \mathbf{Q} \succ 0.$$

This $\mathbf{P}$ exists because $\mathbf{A}_m$ is Hurwitz. Define the composite Lyapunov function from §11.3:

$$V(\mathbf{x}, \tilde{\boldsymbol{\theta}}) \;=\; \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x} \;+\; \tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\tilde{\boldsymbol{\theta}}.$$

Compute $\dot{V}$ along trajectories, using $\dot{\tilde{\boldsymbol{\theta}}} = -\dot{\hat{\boldsymbol{\theta}}}$ (since $\boldsymbol{\theta}$ is constant):

$$\begin{aligned}
\dot{V} &= 2\mathbf{x}^{\mathrm{T}}\mathbf{P}\dot{\mathbf{x}} + 2\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\dot{\tilde{\boldsymbol{\theta}}} \\
&= 2\mathbf{x}^{\mathrm{T}}\mathbf{P}\big(\mathbf{A}_m\mathbf{x} + \mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}}\big) - 2\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\dot{\hat{\boldsymbol{\theta}}} \\
&= -\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + 2\mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}} - 2\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\dot{\hat{\boldsymbol{\theta}}}.
\end{aligned}$$

The two cross terms are the design lever. Choose the *adaptive*

*law*

$$\boxed{\dot{\hat{\boldsymbol{\theta}}} \;=\; \boldsymbol{\Gamma}\,\mathbf{Y}(\mathbf{x})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{x}.}$$

Then the cross terms cancel exactly (this is a one-line computation using the fact that scalar-valued bilinear forms commute under transpose), and

$$\dot{V} \;=\; -\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} \;\leq\; 0.$$

**What this gives us.** $\dot{V} \leq 0$ implies $V$ is non-increasing along trajectories, which implies both $\mathbf{x}$ and $\tilde{\boldsymbol{\theta}}$ are uniformly bounded. The plant state and the parameter error stay in a bounded set determined by the initial conditions.

A slightly stronger argument, using Barbalat's lemma[1], gives the stronger conclusion that $\mathbf{x}(t) \to \mathbf{0}$ asymptotically. The plant tracks zero in the closed loop.

The parameter error $\tilde{\boldsymbol{\theta}}$, however, only stays bounded; it does not necessarily converge to zero. The Lyapunov function may equilibrate at some positive value $V(\infty) > 0$ where $\mathbf{x} = \mathbf{0}$ but $\tilde{\boldsymbol{\theta}} \neq \mathbf{0}$. The adaptive law cannot force $\tilde{\boldsymbol{\theta}} \to \mathbf{0}$ without additional structure on the regressor $\boldsymbol{\phi}(\mathbf{x})$. That additional structure is *persistency of excitation*, which Chapter 12 develops.

[1] Barbalat's lemma: if $f : [0,\infty) \to \mathbb{R}$ is uniformly continuous and $\int_0^\infty f(t)\,dt$ exists and is finite, then $f(t) \to 0$ as $t \to \infty$. Applied here with $f(t) = \mathbf{x}(t)^{\mathrm{T}}\mathbf{Q}\mathbf{x}(t) \geq 0$ and using that $\int_0^\infty f \leq V(0) - V(\infty) < \infty$, we get $\mathbf{x}(t) \to \mathbf{0}$.

§ 11.6

## *What Lyapunov methods cannot do*

The Lyapunov method is powerful, but it is structurally limited in three ways.

**First: no parameter convergence without PE.** The composite Lyapunov function proves boundedness of the parameter error but not its convergence to zero. Parameter convergence requires an excitation condition on the regressor that depends on the realized trajectory, not on the schema. We will see this in Chapter 12 §12.4.

**Second: conservatism.** The Lyapunov bounds are sufficient conditions for stability, not necessary ones. A system can be stable without admitting any obvious Lyapunov function; conversely, the constructed $V$ may overstate the stability margin and produce a conservative ultimate bound. The ratio of the Lyapunov-derived bound to the true system response can be arbitrarily large for the same closed loop. This is a real limitation in practice: a controller that satisfies a tight Lyapunov bound may be over-aggressive; a controller designed for the realized response (without Lyapunov machinery) may be more efficient but harder to certify.

**Third: the matching condition itself.** As §11.4 discussed, the matching condition is a structural assumption about how uncertainty enters the dynamics. Real systems often violate it. When matching fails, the basic Lyapunov-adaptive machinery fails, and more sophisticated tools (sliding-mode control, $L_1$ adaptive control, robust adaptive laws with sigma modification) are needed. The framework's approach is to combine matching-based adaptation (where it applies) with curvature-modulated robust control (which does not require matching) and let the

substrate switch between them based on the local curvature obstruction. This is the architectural commitment ICARUS implements; the details are in Chapter 20.

§ 11.7

## *What you take from this chapter*

Three things.

**First.** Lyapunov's method concludes stability of a dynamical system from the existence of a scalar function $V(\mathbf{x})$ that is positive-definite and non-increasing along trajectories. The level sets of $V$ are nested invariant regions; the trajectory's $V$-level cannot increase. The conclusion is reached without solving the differential equation.

**Second.** For adaptive systems, the composite Lyapunov function $V(\mathbf{x}, \tilde{\boldsymbol{\theta}}) = \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x} + \tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\tilde{\boldsymbol{\theta}}$ couples the plant state and the parameter error into a single scalar function on a higher-dimensional manifold. Under the matching condition, the adaptive law $\dot{\hat{\boldsymbol{\theta}}} = \boldsymbol{\Gamma}\,\mathbf{Y}(\mathbf{x})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ produces $\dot{V} = -\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} \leq 0$, which gives bounded $\tilde{\boldsymbol{\theta}}$ and asymptotic $\mathbf{x} \to \mathbf{0}$.

**Third.** The matching condition—that uncertain parameters enter the dynamics through the same channel as the control input—is the structural assumption that makes direct adaptive feedback possible. It is restrictive but covers a useful design envelope. When matching fails, the framework's curvature-modulated alternative becomes the right tool, and the substrate lets the architecture switch between them.

Chapter 12 takes this machinery and specializes it to model-reference adaptive control, the canonical textbook adaptive law.

The persistency-of-excitation condition that determines when parameter convergence happens—and why ICARUS monitors it at runtime rather than checking it at compile time—is the chapter's central practical theme.

*Vamos.*

Chapter 12

# *Model-Reference Adaptive Control*

*The fundamental purpose of an adaptive controller is to compensate online for parameter uncertainty using only measurement data.*
*Petros A. Ioannou and Jing Sun, Robust Adaptive Control (Prentice Hall, 1995), Preface*

## § 12.0 *The MIT rule, and what came after*

The kid borrows a friend's bike. The friend is a year older; the bike is bigger; the handlebars sit further forward, the saddle is taller, the chain ring has a couple more teeth. She gets on and starts riding. The first thirty seconds feel wrong. Her muscles are producing responses calibrated to her own bike, and the new bike answers them slightly off—a little more roll than she expected, the brake reaching a little later, the gearing pulling harder under load. She does not stop riding. She rides, watches the gap between what she expects to happen and what does

happen, and quietly adjusts. Her grip tightens or loosens, her lean changes by a degree, her timing on the pedals shifts. By the time she has ridden a block her adjustments have settled and the new bike feels like a bike she has been riding for years.

The control-theory name for what the kid just did is **model-reference adaptive control**—MRAC. Before any math, the picture in plain English:

- *Reference model.* The kid carries, in her body's memory, a sense of how a bike *should* respond to her inputs. That sense is the reference model: a description of the closed-loop behavior she wants. The actual plant (the new bike) does not match the reference exactly, but she has a clean target to aim at.
- *Tracking error.* The difference between what the new bike actually does and what the reference says it should do. The kid feels this as "the bike answered a little differently than I expected."
- *Adaptive law.* The rule the kid is using, unconsciously, to adjust her muscle responses based on the tracking error. The control-theory version is mathematical: a rule that says how to update the controller's gains as a function of the error signal.
- *Regressor.* The piece of the state the adaptive law watches. For the kid, it might be "how the bike rolls when I shift weight"—a specific input-output relationship. For a formal controller, it is a vector $\boldsymbol{\phi}(\mathbf{x})$ of state features that participate in the dynamics being tuned.
- *The matching condition.* A structural requirement: the controller's adjustable gains have to be able to actually compensate for whatever is different between the new bike

and the reference. If the new bike has a feature (say, a flat tire) that no muscle adjustment can fix, MRAC will not save the kid; she needs to get off.

**Where the formal version came from.** MRAC was invented at MIT in the late 1950s by a group around H. P. Whitaker working on self-optimizing flight control for high-performance aircraft. The original adjustment rule—known as the *MIT rule*, $\dot{\hat{\boldsymbol{\theta}}} = -\gamma\, \boldsymbol{\phi}(\mathbf{x})\, e$ in the notation we will use throughout this chapter—was gradient descent on the squared tracking error. It was elegant and intuitive. It was also, in many flight-test implementations through the early 1960s, *unstable*: gain settings that worked at one flight condition diverged at another, and the controller would converge on the test bench and then destabilize the aircraft in flight. The X-15 program ran into this; multiple service programs ran into it. The kid's biological version of MRAC works because her body's adjustment rule is already Lyapunov-stable in some implicit sense (millions of years of evolutionary tuning); the first formal flight-test version was not, and the aircraft paid for the difference.

The fix came from P. C. Parks in 1966[1], who re-derived the adaptive law from a Lyapunov stability argument rather than from gradient descent on the squared error. The redesigned law—essentially the one Chapter 11 derived from a composite Lyapunov function on the joint state-and-parameter manifold—was provably stable (bounded closed-loop signals, asymptotic tracking under the matching condition). Modern MRAC dates from Parks's redesign.

This chapter develops MRAC in its post-Parks form. The previous chapter set up the abstract machinery—Lyapunov functions,

[1] Parks, P.C. "Lyapunov Redesign of Model Reference Adaptive Control Systems," *IEEE Transactions on Automatic Control* AC-11(3), 362–367 (1966). The paper formalized what had been informal practice in the post-MIT-rule debugging years.

ultimate boundedness, the matching condition, the composite Lyapunov function on the joint state-and-parameter manifold. This chapter walks through what those tools *do* on a specific problem, with intuition before formalism at every step. The kid on the borrowed bike is what the math is about.

§ 12.1

## *What adaptive control is doing*

Before any equations, here is what we are trying to do.

You have a vehicle—let's keep it concrete and say a small quadcopter, or a 1-DOF cart on a track, or a satellite reaction wheel—whose dynamics depend on parameters you do not know exactly. Maybe the mass changes during flight (fuel burn, payload release). Maybe a drag coefficient depends on atmospheric conditions you cannot measure. Maybe a motor's effectiveness has drifted from its factory-calibrated value. In each case, the dynamics equation contains a term you cannot evaluate from first principles because you do not know one of its inputs.

You want the closed-loop system to behave a specific way regardless of what that unknown parameter happens to be. "Behave a specific way" is going to be made precise by a *reference model*: a mathematical specification of the target closed-loop dynamics, written down by the designer at design time. The reference model says, in formal language: *here is the response curve I want to see from this vehicle when the pilot moves the stick this way.* The reference model is not the actual vehicle. It is the vehicle the designer would like to be flying. It has the desired bandwidth, damping, settling time, overshoot.

The adaptive controller's job is to make the actual vehicle's response approach the reference model's response over time, while

learning the unknown parameter from the difference between the two.

**Every kid who has ever ridden a borrowed bike does MRAC for the first thirty seconds.** Think about what happens when you get on a bicycle that is not yours. You do not know the brake response (how hard do you have to squeeze before the wheel slows?). You do not know the gear ratio (how fast does the pedal effort translate into wheel speed?). You do not know the bike's total mass (your own plus whatever is on the rack). For the first half-block you ride cautiously, testing each control: small brake squeeze, small pedal push, small handlebar turn. You watch how the bike responds. Each test refines your internal model of the bike's parameters. After thirty seconds you have estimated everything well enough to ride at full speed—you have completed a small MRAC adaptation, with the "reference model" being your prior of what a bike should feel like, and the regressor being your test inputs.

That is what we are about to formalize. The kid in §1.0 had done this on enough bikes that she had a strong intuitive prior for what bikes felt like—which was why she could see immediately, without testing, that the lawnmower-bike configuration was outside the regime where her learned model applied. The MRAC machinery in this chapter is the formal version of what your body already does on any new vehicle: test, observe, estimate, refine, ride.

**The running example for this chapter.** To keep the math concrete, we will work the entire chapter on the following scalar system—the same system Lab III implements:

$$\dot{x} \;=\; a\,x \;+\; b(u + \theta\,\phi(x)), \qquad \phi(x) \;=\; x|x|.$$

The state $x$ is a velocity. The known coefficients $a = -0.5$ and $b = 1$ are the linear drag and the control effectiveness. The

function $\phi(x) = x|x|$ is the quadratic-drag form. The coefficient $\theta$ multiplying $\phi$ is the *unknown* parameter we have to estimate—a quadratic-drag coefficient that depends on the vehicle's shape and the fluid's density, which we will pretend we cannot measure offline.

For the chapter's exposition we will say the truth is $\theta = 0.4$, but the controller does not know that; it has to figure it out from data. The reference model will be a faster first-order system:

$$\dot{x}_m \;=\; a_m\, x_m \;+\; b_m\, r(t), \qquad a_m = -2, \quad b_m = 2,$$

where $r(t)$ is the pilot's stick command. The closed-loop dynamics we want is a stable first-order response with time constant $1/|a_m| = 0.5$ seconds. The actual plant, with its unknown drag term, is slower (open-loop time constant $1/|a| = 2$ seconds) and nonlinear; the adaptive controller has to make it behave faster and linear, even without knowing $\theta$.

§ 12.2

## *The plant, the reference model, and the control law*

With the running example in hand, let us write down the full setup carefully. We will state things abstractly (vector notation, multiple states and controls) for generality, but every step will reduce to the scalar example.

**The plant.** An $n$-dimensional state $\mathbf{x} \in \mathbb{R}^n$, an $m$-dimensional control $\mathbf{u} \in \mathbb{R}^m$, and a $d$-dimensional unknown constant parameter vector $\boldsymbol{\theta} \in \mathbb{R}^d$ that enters through a known regressor $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^d$:

$$\dot{\mathbf{x}} \;=\; \mathbf{A}\,\mathbf{x} \;+\; \mathbf{B}\big(\mathbf{u} + \mathbf{Y}(\mathbf{x})\,\boldsymbol{\theta}\big).$$

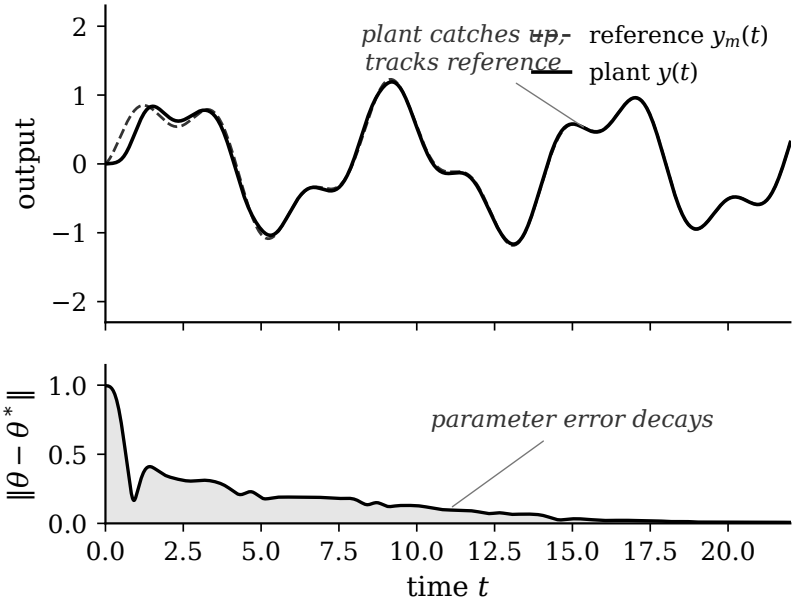



Figure 12.1: MRAC tracking convergence. Top: the reference-model output $y_m(t)$ (dashed) is the response we want; the plant output $y(t)$ (solid) starts mis-tracking because its controller parameters are wrong, then catches up as the adaptive law tunes them. Bottom: the parameter error $\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|$ decays toward zero, the signature of successful adaptation under a persistently exciting reference. Without that excitation the error stays bounded but does not converge—a distinction made precise in §12.6.

Let us unpack what each piece is:

- $\mathbf{Ax}$: the *known* linear part of the dynamics. For our scalar example, this is $a\,x = -0.5\,x$—the linear drag we know about.
- $\mathbf{Bu}$: the *known* control input channel. The control input $\mathbf{u} \in \mathbb{R}^m$ enters through the matrix $\mathbf{B} \in \mathbb{R}^{n\times m}$ (a column vector $b = 1$ in our scalar example). The matched uncertainty $\mathbf{Y}(\mathbf{x})\,\boldsymbol{\theta}$ enters through the same $\mathbf{B}$ channel, with $\mathbf{Y}(\mathbf{x}) \in \mathbb{R}^{m\times d}$ the matrix-valued regressor and $\boldsymbol{\theta} \in \mathbb{R}^d$ the unknown parameter vector. In the SISO case ($m = 1$), $\mathbf{Y}(\mathbf{x})$ is the row vector $\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}$ and the product reduces to the more common $\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$ form.
- $\mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\boldsymbol{\theta}$: the *unknown* nonlinear part. The function $\boldsymbol{\phi}(\mathbf{x})$ is known—in our example it is $x|x|$, the shape of the quadratic-drag term. The coefficient vector $\boldsymbol{\theta}$ is what we do not know. *Critically*, this term enters through the same $\mathbf{B}$ as the control. That is the *matching condition* from Chapter 11 §11.4: the uncertain dynamics is reachable from the control channel.

Why does the matching condition matter? Because it means the controller can, in principle, cancel the uncertain term directly. If the controller knew $\boldsymbol{\theta}$ exactly, it could send the input $\mathbf{u} = -\mathbf{Y}(\mathbf{x})\,\boldsymbol{\theta} + (\text{rest of the control law})$, and the uncertain term would vanish from the closed-loop dynamics. The controller does not know $\boldsymbol{\theta}$, so it uses an *estimate* $\hat{\boldsymbol{\theta}}$ in its place. The adaptive law's job is to make $\hat{\boldsymbol{\theta}}$ converge to $\boldsymbol{\theta}$ over time.

**The reference model.** The designer specifies the target dynamics:

$$\dot{\mathbf{x}}_m \;=\; \mathbf{A}_m\,\mathbf{x}_m \;+\; \mathbf{B}_m\,\mathbf{r}(t).$$

Here:

- $\mathbf{x}_m \in \mathbb{R}^n$ is the *reference state*—what we wish the plant's state could look like.
- $\mathbf{A}_m$ is a known Hurwitz matrix—the closed-loop poles we want, chosen for the right damping and bandwidth.
- $\mathbf{B}_m$ is the reference-input matrix—how the pilot's command $\mathbf{r}(t)$ couples into the reference state.
- $\mathbf{r}(t)$ is the bounded reference signal (the pilot's stick, the autopilot's commanded waypoint, whatever is driving the system).

For our scalar example, $a_m = -2$, $b_m = 2$. The reference model says: when the pilot pushes the stick to $r = 1$, the target velocity is $x_m \to 1$ with first-order rise time $\sim 0.5$ seconds. That is the response we want the actual plant to produce.

**The control law.** The adaptive controller's input to the plant has three pieces:

$$\boxed{\mathbf{u}(t) \;=\; \mathbf{K}_x^{\mathrm{T}}\mathbf{x}(t) \;+\; \mathbf{K}_r^{\mathrm{T}}\mathbf{r}(t) \;-\; \mathbf{Y}\big(\mathbf{x}(t)\big)\,\hat{\boldsymbol{\theta}}(t).}$$

Each term has a job:

1. $\mathbf{K}_x^{\mathrm{T}}\mathbf{x}$ — the *feedback gain*. This term places the closed-loop poles at the reference-model locations. It is the part of the control law that says "act on the current state to push it where it should go."
2. $\mathbf{K}_r^{\mathrm{T}}\mathbf{r}$ — the *feedforward gain*. This term injects the reference signal so that the closed-loop responds to $\mathbf{r}$ with the same gain as the reference model. Without it, the closed-loop poles would be right but the steady-state gain would be wrong.
3. $-\mathbf{Y}(\mathbf{x})\,\hat{\boldsymbol{\theta}}$ — the *adaptive feedforward*. This term is the controller's attempt to cancel the unknown uncertainty

term using its current best estimate $\hat{\boldsymbol{\theta}}$. As $\hat{\boldsymbol{\theta}}$ converges to $\boldsymbol{\theta}$, this term cancels the uncertainty more and more accurately.

**The matching gains.** The fixed gains $\mathbf{K}_x$ and $\mathbf{K}_r$ are not adapted—they are computed once, at design time, from a single requirement: the closed-loop nominal dynamics (ignoring the uncertainty term for a moment) should match the reference model. That requirement reads

$$\mathbf{A} + \mathbf{B}\mathbf{K}_x^{\mathrm{T}} \;=\; \mathbf{A}_m, \qquad \mathbf{B}\mathbf{K}_r^{\mathrm{T}} \;=\; \mathbf{B}_m.$$

The first equation says: the closed-loop $\mathbf{A}$-matrix under feedback $\mathbf{K}_x$ should equal the reference-model $\mathbf{A}_m$. The second says: the closed-loop $\mathbf{B}$-matrix under feedforward $\mathbf{K}_r$ should equal the reference $\mathbf{B}_m$.

Two distinct conditions are needed for this to have a solution, and the literature sometimes blurs them:

- *Uncertainty matching* (§11.4): the uncertain term $\mathbf{Y}(\mathbf{x})\,\boldsymbol{\theta}$ enters through the same $\mathbf{B}$ channel as $\mathbf{u}$. This is what lets the controller cancel the uncertainty with $-\mathbf{Y}(\mathbf{x})\,\hat{\boldsymbol{\theta}}$.
- *Reference-model matching*: there exist gains $\mathbf{K}_x^*$ and $\mathbf{K}_r^*$ such that $\mathbf{A} + \mathbf{B}\mathbf{K}_x^{*\mathrm{T}} = \mathbf{A}_m$ and $\mathbf{B}\mathbf{K}_r^{*\mathrm{T}} = \mathbf{B}_m$. This is a structural reachability/realizability condition on the pair $(\mathbf{A}_m, \mathbf{B}_m)$ relative to $(\mathbf{A}, \mathbf{B})$; it does not follow from uncertainty matching alone.

Both conditions are required: uncertainty matching to enable the Lyapunov cancellation, reference-model matching to ensure the target $(\mathbf{A}_m, \mathbf{B}_m)$ dynamics are reachable from the plant's $(\mathbf{A}, \mathbf{B})$ by static feedback.

For our scalar example:

$$K_x \;=\; \frac{a_m - a}{b} \;=\; \frac{-2 - (-0.5)}{1} \;=\; -1.5, \qquad K_r \;=\; \frac{b_m}{b} \;=\; 2.$$

The feedback gain $K_x = -1.5$ pulls the closed-loop pole from $a = -0.5$ to $a_m = -2$. The feedforward gain $K_r = 2$ matches the reference-model's DC gain. These are fixed, computed once.

The adaptive feedforward $-\hat{\theta}\,\phi(x)$ does the rest—it estimates the unknown $\theta$ online and uses the estimate to cancel the quadratic-drag term in real time.

§ 12.3

## *Deriving the closed-loop tracking-error dynamics*

Now substitute the control law into the plant equation and see what closed-loop dynamics result. This is where the matching condition pays off algebraically.

Start with the plant:

$$\dot{\mathbf{x}} \;=\; \mathbf{A}\mathbf{x} \;+\; \mathbf{B}\big(\mathbf{u} + \mathbf{Y}(\mathbf{x})\,\boldsymbol{\theta}\big).$$

Substitute the control law $\mathbf{u} = \mathbf{K}_x^{\mathrm{T}}\mathbf{x} + \mathbf{K}_r^{\mathrm{T}}\mathbf{r} - \mathbf{Y}(\mathbf{x})\,\hat{\boldsymbol{\theta}}$:

$$\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\big(\mathbf{K}_x^{\mathrm{T}}\mathbf{x} + \mathbf{K}_r^{\mathrm{T}}\mathbf{r} - \mathbf{Y}(\mathbf{x})\,\hat{\boldsymbol{\theta}} + \mathbf{Y}(\mathbf{x})\,\boldsymbol{\theta}\big) \\
&= \underbrace{\big(\mathbf{A} + \mathbf{B}\mathbf{K}_x^{\mathrm{T}}\big)}_{=\mathbf{A}_m}\mathbf{x} + \underbrace{\mathbf{B}\mathbf{K}_r^{\mathrm{T}}}_{=\mathbf{B}_m}\mathbf{r} + \mathbf{B}\,\mathbf{Y}(\mathbf{x})\big(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\big) \\
&= \mathbf{A}_m\mathbf{x} + \mathbf{B}_m\mathbf{r} + \mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}},
\end{aligned}$$

where I have introduced $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} - \hat{\boldsymbol{\theta}}$ for the *parameter error*—the difference between the true parameter and the current estimate.

Look at what just happened. The plant's closed-loop dynamics under our control law are:

$$\dot{\mathbf{x}} \;=\; \mathbf{A}_m\mathbf{x} + \mathbf{B}_m\mathbf{r} + \mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}}.$$

The first two terms are *exactly the reference model's right-hand side*. If the parameter error $\tilde{\boldsymbol{\theta}}$ were zero, the plant would satisfy the same ODE as the reference model, with the same initial-condition response. The plant would track the reference model exactly.

The third term, $\mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}}$, is the disturbance from the parameter error. As long as the adaptive law drives $\tilde{\boldsymbol{\theta}}$ toward zero, this disturbance shrinks, and the plant's trajectory converges to the reference model's trajectory.

**The tracking-error dynamics.** Subtract the reference model from the plant. Let $\mathbf{e} = \mathbf{x} - \mathbf{x}_m$ be the tracking error:

$$\begin{aligned}\dot{\mathbf{e}} &= \dot{\mathbf{x}} - \dot{\mathbf{x}}_m \\ &= \mathbf{A}_m\mathbf{x} + \mathbf{B}_m\mathbf{r} + \mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}} - (\mathbf{A}_m\mathbf{x}_m + \mathbf{B}_m\mathbf{r}) \\ &= \mathbf{A}_m\mathbf{e} + \mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}}.\end{aligned}$$

The reference signal $\mathbf{r}$ has cancelled out entirely. The tracking error's dynamics depend only on:

- The reference-model's $\mathbf{A}_m$ (which is Hurwitz—tracking error decays naturally toward zero in the absence of parameter error).
- The disturbance $\mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}}$, which is what the adaptive law has to kill.

This is the cleanest closed-loop equation you can hope for. The matching condition—the structural assumption that uncertainty enters through the input channel—is what lets this collapse happen. Without matching, the disturbance term would enter through a different matrix and the algebra would not close on a single $\mathbf{B}$ multiplying the parameter error. With matching, the closed loop reduces to: "reference-model dynamics, plus a disturbance proportional to the parameter error."

The adaptive law's job is now clear: drive $\tilde{\boldsymbol{\theta}}$ to zero so that the disturbance vanishes and the tracking error decays under $\mathbf{A}_m$.

§ 12.4

## *The adaptive law, derived from Lyapunov*

We need an update rule for $\hat{\boldsymbol{\theta}}$ that drives $\tilde{\boldsymbol{\theta}}$ toward zero. The naive choice—gradient descent on the squared tracking error—was the MIT rule of §12.0, and it failed in flight test. The Lyapunov-derived alternative, due to Parks 1966, is the one we will use.

The construction is the composite-Lyapunov-function machinery of Chapter 11 §11.3, specialized to this problem. Recall the construction: define a scalar function $V$ on the joint state $(\mathbf{e}, \tilde{\boldsymbol{\theta}}) \in \mathbb{R}^n \times \mathbb{R}^d$ that is positive-definite on the joint state, choose the adaptive law to make $\dot{V}$ negative semi-definite, and conclude boundedness from Lyapunov's theorem.

**Step 1: pick the Lyapunov function.** Try the composite quadratic form

$$V(\mathbf{e}, \tilde{\boldsymbol{\theta}}) \;=\; \mathbf{e}^{\mathrm{T}}\mathbf{P}\mathbf{e} \;+\; \tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\tilde{\boldsymbol{\theta}},$$

where:

- $\mathbf{P} \succ 0$ is a positive-definite matrix to be determined by the algebraic Lyapunov equation associated with $\mathbf{A}_m$.
- $\boldsymbol{\Gamma} \succ 0$ is a positive-definite matrix that the designer chooses—it sets the adaptation gain (more on this in §12.5).

This is positive-definite on the joint state because both summands are positive-definite in their respective arguments. The first term penalizes tracking error; the second penalizes parameter error. $V$ is zero only when both errors are zero.

**Step 2: compute $\dot{V}$ along the closed-loop trajectory.** Differentiate $V$ with respect to time, using the tracking-error dynamics $\dot{\mathbf{e}} = \mathbf{A}_m\mathbf{e} + \mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}}$ and the parameter-error dynamics $\dot{\tilde{\boldsymbol{\theta}}} = -\dot{\hat{\boldsymbol{\theta}}}$ (the true $\boldsymbol{\theta}$ is constant, so its derivative is zero):

$$\begin{aligned}\dot{V} &= 2\mathbf{e}^{\mathrm{T}}\mathbf{P}\dot{\mathbf{e}} \;+\; 2\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\dot{\tilde{\boldsymbol{\theta}}} \\ &= 2\mathbf{e}^{\mathrm{T}}\mathbf{P}\left(\mathbf{A}_m\mathbf{e} + \mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}}\right) - 2\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\dot{\hat{\boldsymbol{\theta}}}.\end{aligned}$$

The first term expands using the algebraic Lyapunov equation. Choose $\mathbf{P}$ to satisfy

$$\mathbf{A}_m^{\mathrm{T}}\mathbf{P} + \mathbf{P}\mathbf{A}_m \;=\; -\mathbf{Q}$$

for some $\mathbf{Q} \succ 0$ (a designer's choice; for our scalar example, with $a_m = -2$ and choosing $Q = 1$, we get $P = -Q/(2a_m) = 0.25$). This $\mathbf{P}$ exists because $\mathbf{A}_m$ is Hurwitz. Then $2\mathbf{e}^{\mathrm{T}}\mathbf{P}\mathbf{A}_m\mathbf{e} = \mathbf{e}^{\mathrm{T}}(\mathbf{A}_m^{\mathrm{T}}\mathbf{P} + \mathbf{P}\mathbf{A}_m)\mathbf{e} = -\mathbf{e}^{\mathrm{T}}\mathbf{Q}\mathbf{e}$, and so:

$$\dot{V} \;=\; -\mathbf{e}^{\mathrm{T}}\mathbf{Q}\mathbf{e} \;+\; 2\mathbf{e}^{\mathrm{T}}\mathbf{P}\mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}} \;-\; 2\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\dot{\hat{\boldsymbol{\theta}}}.$$

The first term is what we want—it is negative-definite in $\mathbf{e}$. The remaining two terms are the design knobs. We want them to cancel.

**Step 3: choose the adaptive law to cancel the cross terms.** The two remaining terms both involve $\tilde{\boldsymbol{\theta}}$. The first is $2\mathbf{e}^{\mathrm{T}}\mathbf{P}\mathbf{B}\,\mathbf{Y}(\mathbf{x})\,\tilde{\boldsymbol{\theta}}$, which is a scalar (so it equals its transpose, $2\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\mathbf{Y}(\mathbf{x})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{e}$). The second is $-2\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\Gamma}^{-1}\dot{\hat{\boldsymbol{\theta}}}$.

For these two to cancel, we need

$$\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\big(\mathbf{Y}(\mathbf{x})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{e} - \boldsymbol{\Gamma}^{-1}\dot{\hat{\boldsymbol{\theta}}}\big) \;=\; 0.$$

This must hold for *all* $\tilde{\boldsymbol{\theta}}$ (we do not know what it is in advance), so the parenthesized expression must be zero:

$$\dot{\hat{\boldsymbol{\theta}}} \;=\; \boldsymbol{\Gamma}\,\boldsymbol{\phi}(\mathbf{x})\,\mathbf{B}^{\mathrm{T}}\mathbf{P}\,\mathbf{e}(t).$$

This is the *MRAC adaptive law*. With this choice, the two cross terms cancel exactly, and:

$$\boxed{\dot{V} \;=\; -\mathbf{e}^{\mathrm{T}}\mathbf{Q}\mathbf{e} \;\le\; 0.}$$

**Step 4: read the conclusion.** Since $\dot{V} \le 0$, the Lyapunov function $V$ is non-increasing along trajectories. Since $V \ge 0$ and is bounded below by zero, and is non-increasing, it converges to some finite limit $V(\infty) \ge 0$. This means both $\mathbf{e}$ and $\tilde{\boldsymbol{\theta}}$ are bounded (because $V$ is the sum of their quadratic forms).

A slightly stronger argument (Barbalat's lemma[2]) upgrades this to $\mathbf{e}(t) \to \mathbf{0}$ asymptotically.

So under the MRAC adaptive law:

- The plant's tracking error converges to zero. The plant tracks the reference model asymptotically.
- The parameter error stays bounded.
- The parameter error does *not* necessarily converge to zero. We get bounded $\tilde{\boldsymbol{\theta}}$, but unless additional structure holds, $\hat{\boldsymbol{\theta}}$ may stop evolving before it reaches $\boldsymbol{\theta}$. (That additional structure is persistency of excitation, in §12.6.)

[2]Barbalat's lemma: if $f : [0,\infty) \to \mathbb{R}$ is uniformly continuous and $\int_0^\infty f(t)\,dt$ exists and is finite, then $f(t) \to 0$ as $t \to \infty$. Applied here with $f(t) = \mathbf{e}(t)^{\mathrm{T}}\mathbf{Q}\mathbf{e}(t) \ge 0$ and using that $\int_0^\infty f \le V(0) - V(\infty) < \infty$, we get $\mathbf{e}(t) \to \mathbf{0}$.

The third point is operationally important and we will come back to it.

**Specialized to our scalar example.** With $a_m = -2$, $b = 1$, $P = 0.25$, $\mathbf{\Gamma} = \Gamma = 5$ (a scalar), $\phi(x) = x|x|$, the MRAC adaptive law becomes

$$\dot{\hat{\theta}} \;=\; \Gamma\,\phi(x)\,b\,P\,e \;=\; 5 \cdot x|x| \cdot 1 \cdot 0.25 \cdot e \;=\; 1.25\,x|x|\,e.$$

The estimate $\hat{\theta}$ increases when $x|x|\,e > 0$ (the plant is moving in the direction where the quadratic drag is producing additional disturbance against the reference) and decreases otherwise. The update is proportional to the product of the regressor's value and the tracking error—which is exactly the gradient-descent direction on the squared tracking error in the right metric.

§ 12.5

## *Why this is gradient descent on the parameter manifold*

The adaptive law $\dot{\hat{\boldsymbol{\theta}}} = \mathbf{\Gamma}\,\boldsymbol{\phi}(\mathbf{x})\,\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{e}$ has a clean geometric reading.

Treat the parameter space $\Theta = \mathbb{R}^d$ as a smooth manifold. The matrix $\mathbf{\Gamma} \succ 0$ defines a Riemannian metric on this manifold:

$$\langle \delta\boldsymbol{\theta}_1, \delta\boldsymbol{\theta}_2 \rangle_{\mathbf{\Gamma}^{-1}} \;=\; \delta\boldsymbol{\theta}_1^{\mathrm{T}}\mathbf{\Gamma}^{-1}\delta\boldsymbol{\theta}_2.$$

The smaller $\mathbf{\Gamma}$ is, the larger the metric $\mathbf{\Gamma}^{-1}$, the longer distances are on $\Theta$, the slower the adaptive law's gradient steps look. The larger $\mathbf{\Gamma}$, the more aggressive the adaptation. This is the designer's tuning knob: how confident am I that the parameter

is approximately constant? If yes, use small $\mathbf{\Gamma}$ (slow, smooth adaptation). If the parameter may drift in time, use larger $\mathbf{\Gamma}$ (aggressive, responsive adaptation, possibly noisier).

**The natural-gradient reading.** The composite Lyapunov function $V$ has a gradient with respect to $\hat{\boldsymbol{\theta}}$ in the $\mathbf{\Gamma}^{-1}$ metric on $\Theta$. The MRAC adaptive law moves $\hat{\boldsymbol{\theta}}$ in the direction that decreases $V$ as fast as possible in this metric, projected onto the operational subspace defined by the regressor and the closed-loop coupling. In short: the adaptive law is *natural-gradient descent* on $V$ on the parameter manifold $\Theta$, with $\mathbf{\Gamma}$ setting the local geometry.

This is structurally parallel to Chapter 8 §8.5's reading of LQR. There, the LQR closed-loop dynamics turned out to be gradient descent on the value function $V_{\text{LQR}} = \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ on the state manifold, in the $g_{\mathbf{P}}$ metric. Here, the MRAC *adaptation* is gradient descent on the composite Lyapunov function $V$ on the parameter manifold, in the $\mathbf{\Gamma}^{-1}$ metric. Different metrics; same structural pattern. The framework reads both as Riemannian-gradient flows; the choice of metric is a gauge choice that the designer picks.

**What this buys us for the Curvature Floor.** The Riemannian metric $\mathbf{\Gamma}^{-1}$ on $\Theta$ has a curvature in the differential-geometric sense. For time-varying or state-dependent $\mathbf{\Gamma}$, this curvature can be non-zero. The Curvature Floor inequality of Chapter 7 §7.5 includes a term $\beta\,\mathrm{curv}(\bar{\Theta})$ that accounts for this. Chapter 13 will develop the inequality with the curvature term reinstated; for this chapter, the takeaway is that the adaptive law's choice of $\mathbf{\Gamma}$ is a gauge choice on the parameter manifold, and the gauge controls both the adaptation's aggressiveness and the share of the Curvature Floor budget the adaptation consumes.

§ 12.6

## *Persistency of excitation, intuitively and formally*

We have shown that the MRAC adaptive law drives the tracking error to zero asymptotically. We have not shown that the parameter error converges to zero. The reason is that the adaptive law can run out of information.

**The intuitive picture.** Imagine our scalar example with the pilot's command $r(t)$ held constant at, say, $r = 1$. The plant settles into a steady state $x \to x_{\text{ss}}$, the reference model settles into $x_m \to x_{m,\text{ss}}$, and (because the adaptive law has been adjusting $\hat{\theta}$ all along) these two steady states are close—the tracking error $e \to 0$. The regressor $\phi(x) = x|x|$ also settles to a constant, $\phi(x_{\text{ss}})$.

Now look at the adaptive law: $\dot{\hat{\theta}} = 1.25\,\phi(x)\,e$. With $e \to 0$, this update term vanishes. The parameter estimate stops moving. But the plant is still being affected by the term $b\theta\phi(x_{\text{ss}})$—the actual drag at the actual steady state—which the controller is approximately cancelling using $b\hat{\theta}\phi(x_{\text{ss}})$. As long as the steady-state cancellation is good enough, the tracking error stays small. But *the parameter estimate $\hat{\theta}$ has stopped at whatever value it happened to be at when the trajectory entered the steady state.* If that value was 0.39 and the truth was 0.40, the controller is operating with an incorrect $\hat{\theta}$ that is good enough for the current operating point but might be wrong for other points.

**Why this happens structurally.** The adaptive law is gradient descent on the composite Lyapunov function. When the gradient is zero, the descent stops. The gradient is zero whenever $\phi(x) \cdot e = 0$ in our scalar example—which happens when $e = 0$ (the plant tracks the reference model, regardless of whether $\hat{\theta}$ is right). At that point the adaptive law has *no signal* to tell it

whether to keep adjusting $\hat{\theta}$ or stay put.

The only way to keep the gradient nonzero, even when the plant is in steady state, is to keep the regressor changing. If the regressor visits many directions of the parameter space over time, the adaptive law sees diverse update signals and converges to the right answer. If the regressor is stuck at one value, the law converges to *some* answer, not necessarily the right one.

**The formal condition.** The regressor $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^d$ evaluated along the closed-loop trajectory $\mathbf{x}(t)$ is *persistently exciting* (PE) if there exist positive constants $\alpha, T$ such that

$$\int_t^{t+T} \boldsymbol{\phi}(\mathbf{x}(s))\, \boldsymbol{\phi}(\mathbf{x}(s))^{\mathrm{T}}\, ds \;\succeq\; \alpha \mathbf{I} \qquad \text{for all } t \geq 0.$$

The integral is a $d \times d$ matrix (the *regressor autocorrelation matrix* integrated over a window of length $T$). The condition says: this matrix is uniformly bounded below by a positive multiple of the identity over every window of length $T$. Equivalently: the regressor visits all $d$ directions of the parameter space with at least $\sqrt{\alpha}$ of average amplitude over every $T$-window.

For our scalar example, the condition reduces to

$$\int_t^{t+T} \phi(x(s))^2\, ds \;\geq\; \alpha \qquad \text{for all } t,$$

which says the regressor's energy over every $T$-window is bounded below. In plain terms: the regressor cannot stay zero or near-zero for any extended window. The trajectory has to keep visiting states where $\phi(x) = x|x|$ is meaningfully nonzero.

**The convergence theorem.** If the regressor along the closed-loop trajectory is persistently exciting, then under the MRAC adaptive law the parameter error $\tilde{\boldsymbol{\theta}}(t) \to \mathbf{0}$ exponentially fast. With PE, the adaptive law converges; without PE, it can stop short.

**Why PE cannot be checked at compile time.** The PE condition is a statement about the realized trajectory $\mathbf{x}(t)$ as it maps through the regressor $\boldsymbol{\phi}$. Three reasons it depends on the runtime, not the schema:

1. The reference signal $\mathbf{r}(t)$ has to be rich enough to excite the parameter directions. A constant reference excites only one direction (the constant); a slowly-varying reference excites only a few. The adaptive controller cannot pick what the pilot will command.

2. The plant's closed-loop response is what realizes the regressor's trajectory. Different initial conditions and different references produce different realized $\mathbf{x}(t)$, and only along the realized trajectory does PE make sense.

3. The regressor $\boldsymbol{\phi}$ must be *informative*: different states must produce visibly different regressors. This is a property of the trajectory's image under $\boldsymbol{\phi}$. For our scalar example, $\phi(x) = x|x|$ is informative as long as $x$ is not stuck at zero; for a regressor like $\phi(x) = 1$ (constant), no trajectory can be PE.

None of these three is decidable from the system equations alone. PE is a runtime fact about the realized trajectory, and the substrate has to monitor it.

**What the operational story looks like.** An adaptive controller running in the field may be experiencing PE at this moment and not at the next. A pilot in steady cruise produces a regressor that has been at a near-constant value for minutes; the adaptive law's parameter estimates have drifted somewhere and stopped. The closed loop is tracking the (constant) reference, so nothing visible is wrong. Then the pilot commands a maneuver—PE returns, the regressor starts visiting many states, and the

parameter estimates wake up. If they had drifted significantly during the no-PE period, the closed loop sees a transient as they readjust. If they hadn't, no transient.

This is one of the recurring failure modes of adaptive flight control. The cure—and the architectural commitment ICARUS implements—is to monitor PE at runtime and gate the adaptive law accordingly. When PE collapses, freeze the parameter or apply a modification term that prevents drift; when PE returns, allow adaptation again. This is what the sidebar in §12.7 describes.

§ 12.7

## *Sidebar: ICARUS implementation of MRAC*

*The following sidebar is reproduced from the ICARUS architecture documentation.*

> ICARUS's MRAC component implements the adaptive law $\dot{\hat{\mathbf{K}}}/dt = -\boldsymbol{\Gamma} \cdot \mathbf{x} \cdot (\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{e})$ as a streaming GQL update on a controller-parameter bundle. The metric $\boldsymbol{\Gamma}^{-1}$ is stored as a schema-level fiber tensor. **Persistency of excitation is monitored at runtime** via the regressor's autocorrelation matrix (PE is a property of the realized trajectory, not of the schema, and cannot be checked at compile time).

**Architectural unpacking.** The framework's substrate makes the MRAC adaptive law a database operation, not a runtime control-law computation. The controller-parameter bundle holds $\hat{\boldsymbol{\theta}}(t)$; the adaptive law's gradient step is a streaming GQL update against this bundle, applied at the controller's update rate

(typically 400 Hz to match the attitude inner loop). The metric $\mathbf{\Gamma}^{-1}$ is stored as a schema-level fiber tensor, which means it is gauge-invariant under coordinate changes on the parameter manifold and is sealed at the schema layer so the runtime cannot accidentally corrupt it.

The PE monitor runs alongside the adaptive law. It maintains a running estimate of the regressor autocorrelation matrix

$$\mathbf{M}(t) \;=\; \int_{t-T}^{t} \boldsymbol{\phi}(\mathbf{x}(s))\, \boldsymbol{\phi}(\mathbf{x}(s))^{\mathrm{T}}\, ds$$

(typically using an exponential window of decay time $T$). When the minimum eigenvalue $\lambda_{\min}(\mathbf{M}(t))$ falls below a threshold $\alpha_{\min}$, the substrate flags the regressor as *not currently PE* and gates the adaptive law accordingly—either freezing the parameter estimate ($\dot{\hat{\boldsymbol{\theta}}} = 0$) or applying a sigma-modification term ($\dot{\hat{\boldsymbol{\theta}}} = \mathbf{\Gamma}\boldsymbol{\phi}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{e} - \sigma\hat{\boldsymbol{\theta}}$) that biases the estimate back toward zero in the unexcited directions.

The PE-monitoring computation is constant-time and runs on the same thread as the adaptive law. The cost of the monitor is small; the cost of *not* monitoring—of letting the adaptive law drift during long no-PE periods—is the kind of failure mode that has destabilized flight-test adaptive controllers since the 1960s. The substrate keeps this from being possible because the schema declares the PE invariant, and the substrate enforces it.

§ 12.8

## *The framework's reading of MRAC's parameter manifold*

Three structural observations that the rest of Part IV will lean on.

**First: the parameter manifold has curvature.** The metric $\mathbf{\Gamma}^{-1}$ on $\Theta = \mathbb{R}^d$ has a Riemannian curvature in the differential-geometric sense. For constant $\mathbf{\Gamma}$, the manifold is flat. For time-varying or state-dependent $\mathbf{\Gamma}$—common in recursive-least-squares adaptive laws, which update $\mathbf{\Gamma}$ based on observed information geometry—the manifold has non-zero curvature. This curvature appears in the Curvature Floor inequality of Chapter 7 §7.5 as the term $\beta\,\mathrm{curv}(\bar{\Theta})$. Chapter 13 reinstates the term explicitly.

**Second: the joint state-parameter system has a Riemannian structure.** The composite Lyapunov function uses two metrics: $\mathbf{P}$ on the state manifold, $\mathbf{\Gamma}^{-1}$ on the parameter manifold. The pair $(\mathbf{P}, \mathbf{\Gamma})$ together defines a product-Riemannian metric on the joint state-parameter space. The closed-loop dynamics under MRAC are gradient flow on this product manifold, in this product metric. The framework's gauge analysis from Chapter 6 applies: under similarity transformations of either the state or the parameter coordinates, the closed-loop trajectory and the parameter convergence behavior are gauge-invariant, while the specific numerical entries of $\mathbf{P}$, $\mathbf{\Gamma}$, and the gain matrices are gauge-dependent.

**Third: PE is the trajectory's holonomy in the regressor manifold.** Read in the bundle language: the closed-loop trajectory $\mathbf{x}(t)$ is a section of the state bundle; the regressor $\boldsymbol{\phi}$ pushes this section forward into a section of the regressor

manifold. PE says: this pushed-forward section's local holonomy is bounded away from zero over every window of length $T$. When the holonomy collapses (the section is approximately constant), the adaptive law's gradient signal vanishes and the parameter estimate stops updating. This is the framework's reading of PE: *adaptation converges when the trajectory's holonomy in the regressor manifold is sufficient.* When it isn't, the substrate must monitor and modify the adaptation accordingly.

This last point is what makes ICARUS's runtime PE monitoring architecturally interesting. The substrate is not just running the controller; it is reading the trajectory's holonomy in the regressor manifold and adjusting the controller's behavior based on that reading. The same curvature-aware substrate that gates the LQR gain on the predicted trajectory's curvature obstruction also gates the adaptive law on the regressor's runtime holonomy. One substrate, two readings, both geometric, both operational at flight time.

§ 12.9

## *What you take from this chapter*

Four things.

**First.** The MRAC problem is: given a plant with an unknown parameter that enters through the input channel (matching condition), and a reference model that specifies the target closed-loop dynamics, find a controller that drives the plant's response to match the reference model's response while learning the unknown parameter from the data.

**Second.** The control law has three pieces: a fixed feedback gain $\mathbf{K}_x$ that places closed-loop poles, a fixed feedforward gain

$\mathbf{K}_r$ that matches the DC response, and an adaptive feedforward $-\mathbf{Y}(\mathbf{x})\,\hat{\boldsymbol{\theta}}$ that uses the current parameter estimate to cancel the uncertain term. The matching gains $(\mathbf{K}_x, \mathbf{K}_r)$ are computed at design time from $\mathbf{A} + \mathbf{B}\mathbf{K}_x^{\mathrm{T}} = \mathbf{A}_m$ and $\mathbf{B}\mathbf{K}_r^{\mathrm{T}} = \mathbf{B}_m$.

**Third.** The Lyapunov-derived adaptive law is

$$\dot{\hat{\boldsymbol{\theta}}} \;=\; \boldsymbol{\Gamma}\,\boldsymbol{\phi}(\mathbf{x})\,\mathbf{B}^{\mathrm{T}}\mathbf{P}\,\mathbf{e},$$

with $\mathbf{P}$ solving $\mathbf{A}_m^{\mathrm{T}}\mathbf{P} + \mathbf{P}\mathbf{A}_m = -\mathbf{Q}$ and $\boldsymbol{\Gamma}$ a designer-chosen adaptation gain. Under this law, the closed-loop tracking error $\mathbf{e}(t)$ converges to zero asymptotically; the parameter error $\tilde{\boldsymbol{\theta}}$ stays bounded but does not necessarily converge to zero. The adaptive law is gradient descent on the composite Lyapunov function in the $\boldsymbol{\Gamma}^{-1}$ Riemannian metric on the parameter manifold.

**Fourth.** Parameter convergence requires *persistency of excitation*: the regressor along the closed-loop trajectory must visit all directions of the parameter manifold with sufficient amplitude over every window of length $T$. PE is a property of the realized trajectory and cannot be guaranteed at design time. ICARUS monitors PE at runtime via the regressor's autocorrelation matrix and gates the adaptive law when the trajectory loses excitation, which is the architectural protection against the adaptive-controller drift problem that has been the recurring failure mode of adaptive flight control since the 1960s.

Chapter 13 returns to the Curvature Floor inequality of Chapter 7 §7.5 with the adaptive terms ($\alpha\,\mathrm{tr}(\boldsymbol{\Gamma}^{-1})$ and $\beta\,\mathrm{curv}(\bar{\Theta})$) reinstated, completing the inequality form left as a forward pointer in that chapter. The MRAC machinery of this chapter is exactly what those adaptive terms are accounting for.

*Vamos.*

# *Lab III: MRAC, PE, and the runtime holonomy of the regressor.*

*Time: about 45–60 minutes. Materials: Python 3.10 or later with* `numpy`*,* `scipy`*,* `matplotlib`*, and* `dhoom`*.*

Lab I built a static fiber bundle; Lab II flew a closed-loop trajectory and stored it. This lab runs the Lyapunov-derived MRAC adaptive law from Chapter 12 against a plant with one unknown parameter, and walks you through the structural fact that makes ICARUS's runtime PE monitoring necessary: the parameter converges when the reference signal is rich enough, and does not when the reference signal is not. You will run both cases and observe the difference.

By the end you will have implemented the post-Parks MRAC adaptive law, watched it track a reference model under parameter uncertainty, observed the regressor's autocorrelation matrix grow when PE is satisfied and stall when it isn't, and stored the resulting parameter-history bundle as DHOOM. The same regressor-autocorrelation computation you'll write is what ICARUS does in its runtime PE monitor.

**The plant we'll use.** A scalar-state mass-damper with an unknown drag coefficient:

$$\dot{x} \;=\; a\,x \;+\; b(u + \theta\,\phi(x)), \qquad \phi(x) = x|x|.$$

The state $x \in \mathbb{R}$ is a velocity; the drag term $\theta\, x|x|$ is the standard quadratic-drag model with unknown coefficient $\theta$. The matching condition is satisfied: the uncertainty enters through the same channel $b$ as the control. We will pretend $\theta = 0.4$ is the truth and let the adaptive law estimate it.

Step 1

## *Set up*

```
mkdir lab3 && cd lab3
python  # or jupyter notebook
```

Verify dependencies (you should have these from Labs I and II):

```
import numpy as np
import scipy.integrate as integ
import dhoom
```

Step 2

## *The plant, the reference model, the matching gains*

```
# Plant scalars.
# a, b: known. theta_true: the UNKNOWN parameter
# the adaptive law must estimate from data.
a = -0.5
b = 1.0
theta_true = 0.4

# Regressor: phi(x) = x*|x|  (quadratic-drag form)
```

```
def phi(x):
    return x * abs(x)

# Reference model: stable first-order target.
# am: pole (Hurwitz, faster than plant).
am = -2.0
bm = 2.0

# Matching gains: A + B*Kx = Am, B*Kr = Bm.
Kx = (am - a) / b    # = -1.5
Kr = bm / b          # =  2.0
print(f'Kx = {Kx}, Kr = {Kr}')
# Kx = -1.5, Kr = 2.0
```

The control law will be $u = K_x x + K_r r - \hat{\theta}\,\phi(x)$. The first two terms place the closed-loop pole at $a_m$ (assuming we knew $\theta$); the third term is the adaptive feedforward.

### Step 3

## *The composite Lyapunov function and the adaptive law*

The Lyapunov function from Chapter 11 §11.5 is $V = Pe^2 + \Gamma^{-1}\tilde{\theta}^2$ where $e = x - x_m$ is the tracking error and $\tilde{\theta} = \theta - \hat{\theta}$ is the parameter error. With scalars: solve the algebraic Lyapunov equation $2a_m P = -Q$ for $P$, given a choice of $Q$.

```
Q = 1.0
P = -Q / (2 * am)    # = -1.0 / (-4.0) = 0.25
print(f'P = {P}')
# P = 0.25

# Adaptive law gain (parameter-manifold metric).
Gamma = 5.0
```

The MRAC adaptive law is $\dot{\hat{\theta}} = \Gamma\,\phi(x)\,B^T P\,e$. With scalars: $\dot{\hat{\theta}} = \Gamma\,\phi(x)\,b\,P\,e$.

Step 4

*Simulate Case A: PE-rich reference (sinusoidal)*

```
def simulate(reference_func, t_final=30, dt=0.005):
    # Run the closed-loop MRAC system; return history dict.
    t_grid = np.arange(0, t_final, dt)
    n = len(t_grid)

    x = 0.0           # plant state
    xm = 0.0          # reference model state
    theta_hat = 0.0   # starts wrong; true theta = 0.4

    # Running regressor autocorrelation (the PE monitor).
    M = 0.0
    window = 2.0      # seconds
    decay = np.exp(-dt / window)

    hist = {
        't': t_grid,
        'x':         np.zeros(n),
        'xm':        np.zeros(n),
        'e':         np.zeros(n),
        'theta_hat': np.zeros(n),
        'M_pe':      np.zeros(n),
    }

    for i, t in enumerate(t_grid):
        r = reference_func(t)
        e = x - xm
        # Control law (matching feedback + adaptive feedforward).
        u = Kx * x + Kr * r - theta_hat * phi(x)
        # Plant dynamics with the TRUE theta
        # (we know truth in sim; the adaptive law doesn't).
        x_dot  = a * x  + b * (u + theta_true * phi(x))
        xm_dot = am * xm + bm * r
        # Adaptive law: gradient of V on the parameter manifold.
        theta_hat_dot = Gamma * phi(x) * b * P * e
        # PE monitor: windowed regressor autocorrelation.
        M = decay * M + (1 - decay) * phi(x) ** 2

        hist['x'][i]         = x
        hist['xm'][i]        = xm
        hist['theta_hat'][i] = theta_hat
```

```
        hist['M_pe'][i]      = M
        hist['e'][i]         = e

        # Euler step.
        x         += x_dot * dt
        xm        += xm_dot * dt
        theta_hat += theta_hat_dot * dt

    return hist

# Case A: sinusoidal reference, rich enough to be PE.
ref_A = lambda t: np.sin(0.5*t) + 0.4 * np.sin(1.7*t)
hist_A = simulate(ref_A)

th_A  = hist_A['theta_hat'][-1]
rms_A = np.sqrt(np.mean(hist_A['e']**2))
M_A   = hist_A['M_pe'][-1]
print('Case A (PE-rich sinusoid):')
print(f'  theta_hat at t=30s : {th_A:.4f}')
print(f'  true theta         : {theta_true}')
print(f'  tracking error RMS : {rms_A:.4f}')
print(f'  M_pe at t=30s      : {M_A:.4f}')
# Case A (PE-rich sinusoid):
#   theta_hat at t=30s : ~0.39 to 0.41
#   true theta         : 0.4
#   tracking error RMS : ~0.05 to 0.10
#   M_pe at t=30s      : ~0.2 to 0.5  (PE satisfied)
```

Case A: the sinusoidal reference excites the regressor across all amplitudes, $M_{\text{pe}}$ stays positive throughout, and the parameter estimate converges to the true value. Tracking error stays small after a brief transient.

Step 5

*Simulate Case B: not PE (constant reference)*

```
# Case B: brief perturbation then back to zero.
# After t >= 5, ref = 0, x -> 0, phi(x) = x|x| -> 0,
```

```
# and M_pe truly collapses. The failure mode the lab needs to
# demonstrate requires phi(x) -> 0, not just phi(x) constant:
# a constant nonzero phi is still PE for a scalar parameter.
ref_B = lambda t: 1.0 if t < 5.0 else 0.0
hist_B = simulate(ref_B)

th_B  = hist_B['theta_hat'][-1]
rms_B = np.sqrt(np.mean(hist_B['e']**2))
M_B   = hist_B['M_pe'][-1]
print('Case B (constant reference, not PE):')
print(f'  theta_hat at t=30s : {th_B:.4f}')
print(f'  true theta         : {theta_true}')
print(f'  tracking error RMS : {rms_B:.4f}')
print(f'  M_pe at t=30s      : {M_B:.4f}')
# Case B (brief perturbation, then PE fails):
#   theta_hat at t=30s : value frozen at whatever the brief PE
#                        period drove it to (e.g., ~0.2 to 0.4
#                        depending on how long PE was satisfied).
#   true theta         : 0.4
#   tracking error RMS : small after t > 5 (x, xm both -> 0)
#   M_pe at t=30s      : ~0  (PE failed; regressor went to zero)
```

Case B: the brief reference perturbation during $0 < t < 5$ excites $\phi(x) = x|x|$ enough that some parameter learning happens. After $t = 5$, the reference returns to zero, $x \to 0$, $\phi(x) \to 0$, and $M_{\text{pe}}$ truly collapses. The parameter estimate $\hat{\theta}$ stops moving from this point on; whatever value it reached during the brief excitation is what it will hold. Tracking error is small in both regimes (the closed loop tracks zero correctly), but the parameter has not converged to the true value because the data after $t = 5$ contains no information about $\theta$.

This is the operational reality of the PE story. *Tracking and parameter convergence are not the same thing.* A well-tracking adaptive controller can be running with an arbitrarily wrong $\hat{\theta}$, and only the runtime PE monitor reveals it. In Case B, the parameter converges only as far as the brief excitation allows; once the regressor goes to zero, the adaptive law has no signal and the estimate freezes.

Step 6

## *The PE monitor in operation*

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(3, 2, figsize=(12, 8), sharex=True)
cases = [(hist_A, 'A: PE-rich'), (hist_B, 'B: not PE')]
for col, (h, name) in enumerate(cases):
    ax[0, col].plot(h['t'], h['x'],  label='x')
    ax[0, col].plot(h['t'], h['xm'], '--', label='xm')
    ax[0, col].set_title(f'Case {name}')
    ax[0, col].legend()

    ax[1, col].plot(h['t'], h['theta_hat'], label='theta_hat')
    ax[1, col].axhline(theta_true, color='r', ls=':',
                       label='true')
    ax[1, col].legend()
    ax[1, col].set_ylabel('theta_hat')

    ax[2, col].plot(h['t'], h['M_pe'], label='M_pe')
    ax[2, col].axhline(0.05, color='gray', ls=':',
                       label='PE threshold')
    ax[2, col].legend()
    ax[2, col].set_xlabel('t (s)')

plt.tight_layout()
plt.savefig('lab3_pe_compare.png', dpi=120)
```

The bottom row is the PE monitor in operation. In Case A, $M_{\text{pe}}$ rises and stays well above the threshold throughout the run. In Case B, $M_{\text{pe}}$ collapses toward zero after the initial transient and the parameter never converges.

**What ICARUS does with this signal.** When $M_{\text{pe}}$ drops below the configured threshold, ICARUS gates the adaptive law: either freezes the parameter estimate ($\dot{\hat{\theta}} = 0$) or applies a sigma-modification term ($\dot{\hat{\theta}} \mathrel{-}= \sigma\hat{\theta}$) that biases the estimate back toward zero in the unexcited directions. This prevents the long-term parameter drift that would otherwise occur during

extended periods of no PE. Without this gating, the adaptive law's $\hat{\theta}$ wanders, and when PE returns, the closed-loop sees a transient that reflects how far $\hat{\theta}$ drifted in the meantime.

Step 7

## *Store the parameter histories as DHOOM*

```
def to_dhoom(hist, name):
    records = []
    for i in range(len(hist['t'])):
        records.append({
            'k': i,
            't':         float(hist['t'][i]),
            'x':         float(hist['x'][i]),
            'xm':        float(hist['xm'][i]),
            'e':         float(hist['e'][i]),
            'theta_hat': float(hist['theta_hat'][i]),
            'M_pe':      float(hist['M_pe'][i]),
        })
    return {name: records}

data_A = to_dhoom(hist_A, 'case_A_PE_rich')
data_B = to_dhoom(hist_B, 'case_B_not_PE')
dh_A = dhoom.encode(data_A)
dh_B = dhoom.encode(data_B)

with open('case_A.dhoom', 'w') as f:
    f.write(dh_A)
with open('case_B.dhoom', 'w') as f:
    f.write(dh_B)

import json
jA = json.dumps(data_A, separators=(',', ':'))
jB = json.dumps(data_B, separators=(',', ':'))
rA = (1 - len(dh_A) / len(jA)) * 100
rB = (1 - len(dh_B) / len(jB)) * 100
print(f'A:  JSON {len(jA):6d}   DHOOM {len(dh_A):6d}'
      f'   ({rA:.1f}% smaller)')
print(f'B:  JSON {len(jB):6d}   DHOOM {len(dh_B):6d}'
```

```
      f'   ({rB:.1f}% smaller)')
# DHOOM is typically ~60-70% smaller for these bundles.
```

Each record is a section of the bundle: $(k, t, x, x_m, e, \hat{\theta}, M_{\text{pe}})$. The encoder will derive $k@0 + 1$ as arithmetic and $t@0 + 0.005$ as arithmetic, dropping both from the records. The remaining five fields per record carry the simulation's actual content.

## Step 8

## *What you take from this lab*

Three things.

**First.** The MRAC adaptive law $\dot{\hat{\theta}} = \Gamma\, \phi(x)\, b\, P\, e$ tracks a reference model under parameter uncertainty. Under the matching condition (which the quadratic-drag plant satisfies), the closed loop is stable and tracking error converges to zero. The parameter estimate $\hat{\theta}$ also converges *when the reference signal is PE-rich.*

**Second.** Persistency of excitation is a property of the realized trajectory, not the schema. The PE monitor $M(t) \approx \int_{t-T}^{t} \phi^2\, ds$ tells you, at runtime, whether the regressor is currently exciting the parameter manifold. When $M$ collapses, parameter convergence stops; the closed loop may still be tracking, but with an incorrect $\hat{\theta}$ that the adaptive law cannot correct from the current data.

**Third.** ICARUS's substrate runs exactly this PE monitor on the runtime regressor and gates the adaptive law accordingly. The framework's claim—that adaptation converges when the trajectory's holonomy in the regressor manifold is sufficient, and that the substrate must check this at runtime because the schema

cannot—is now something you have measured in your own code, not just read in a chapter.

This closes the Part IV companion lab. Chapter 13 will reinstate the adaptive terms in the Curvature Floor inequality and complete the inequality form left as a forward pointer in Chapter 7 §7.5. The MRAC machinery of this lab is exactly what those adaptive terms are bounding.

*Vamos.*

Chapter 13

# *Robust adaptive laws and the Curvature Floor conjecture*

*Adaptation that drifts is worse than no adaptation at all.*
*Petros A. Ioannou and Petar V. Kokotović, Adaptive Systems with Reduced Models (Springer, 1983), paraphrased from §5.4*

## § 13.0 *What this chapter closes*

The kid is back on her borrowed bike from Chapter 12—the friend's bike that was taller, geared differently, with the handlebars set forward. Within a block she had adapted. Her muscle response had shifted to fit the new bike, the tracking error had settled, and the new bike felt familiar.

The question this chapter is about: *what did that adaptation cost?*

Not in calories. In something more specific: in the closed-loop's ability to do its other job. While she was actively adjusting

her response to fit the new bike, her body was committing some of its attention and reflexes to the adaptation. The disturbance-rejection bandwidth of her riding—her capacity to handle a pothole or a sudden wind gust without losing control—was reduced for the duration. An adaptive controller is more flexible than a fixed one, but the flexibility shows up on the other side of the books as bandwidth loss somewhere in the closed-loop response.

Chapter 7 stated this in formal language. The Bode integral—the conservation law from §7.3—says that $\int_0^\infty \log|S(j\omega)|\,d\omega$ is bounded below by $\pi$ times the sum of the unstable poles, for an LTI plant. When the controller is allowed to adapt, an additional bound enters. §7.5 introduced this as the *Curvature Floor inequality*:

$$\int_0^\infty \log|S(j\omega)|\,d\omega \;\geq\; \pi\sum_{i=1}^{m}\mathrm{Re}(p_i^+) \;+\; \alpha\,\mathrm{tr}(\mathbf{\Gamma}^{-1}) \;+\; \beta\,\mathrm{curv}(\bar{\Theta}),$$

and deferred its adaptive terms to this chapter.

**What the new terms mean, in plain English.** Two extra costs on the right-hand side of the inequality, beyond Bode's classical term:

- $\alpha\,\mathrm{tr}(\mathbf{\Gamma}^{-1})$ is *the cost of having adaptation at all.* $\mathbf{\Gamma}$ is the gain matrix on the MRAC adaptive law from Chapter 12—the matrix that controls how aggressively the controller updates its parameter estimates. Larger $\mathbf{\Gamma}$ means faster adaptation; smaller $\mathbf{\Gamma}$ means slower. Either extreme costs something. The $\mathrm{tr}(\mathbf{\Gamma}^{-1})$ term is the integrated cost across the parameter space of having the law be tunable in the first place.
- $\beta\,\mathrm{curv}(\bar{\Theta})$ is *the cost of a curved parameter manifold.* The parameters being adapted live on a manifold $\Theta$ (Chapter 12

§12.5). If that manifold is flat—the parameters are just real numbers with no inherent geometric structure—the cost is zero. If the manifold is curved—there are constraints, dependencies, or nonlinear couplings between parameters—adapting on it has an integrated curvature cost.

Both terms are non-negative when the adaptive law is running. The interpretation of the limits deserves care:

- *Fast-adaptation limit ($\boldsymbol{\Gamma} \to \infty$):* $\operatorname{tr}(\boldsymbol{\Gamma}^{-1}) \to 0$, so the term vanishes. Fast adaptation costs the closed loop very little of its sensitivity budget—in this limit, the law tracks the true parameter so quickly that the closed loop is essentially the post-convergence LTI loop.
- *Slow-adaptation limit ($\boldsymbol{\Gamma} \to 0^+$):* $\operatorname{tr}(\boldsymbol{\Gamma}^{-1}) \to \infty$, so the term blows up. The adaptive transient lasts longer; the closed-loop sensitivity has to absorb more of the parameter mismatch.
- *No-adaptation limit ($\boldsymbol{\Gamma} = 0$, estimates frozen):* the adaptive law is not running at all, so the additional Curvature Floor terms do not apply. The inequality reduces to the LTI Bode result for the frozen-parameter closed loop, and the $\boldsymbol{\Gamma}^{-1}$ symbol is to be read as "vanishing term" under the convention that no adaptive cost is added when no adaptation is running.

Both Curvature Floor terms appear because the act of adapting widens the sensitivity integral, costing the closed loop some of its disturbance-rejection budget. The kid's body, adapting to the borrowed bike, paid these costs in reduced reflex margin for the duration of the adaptation. The Curvature Floor inequality is what the cost looks like when written down formally.

This chapter has four jobs:

1. Walk what the two adaptive terms mean operationally and where they come from in the proof structure.
2. Walk the *robust adaptive laws*—sigma-modification, e-modification, dead-zones—that practical adaptive controllers use to bound parameter drift during periods without persistency of excitation.
3. Reproduce the ICARUS sidebar with its verbatim disclosure about quantitative fitting not being attempted.
4. Tell you exactly what is conjectural and what is verified, so the framework's claims stay honest.

**The kid's anchor for this chapter.** Go back to §1.0 one more time. The kid in the driveway, looking at the lawnmower-bike, was performing her own version of an adaptive controller. Her "parameters"—the engine's actual responsiveness, the bike's actual mass with her on it, the driveway's actual coefficient of friction—were uncertain. Her "regressor"—what she could test from her current position without committing to the ride—was nearly empty. She had no persistency of excitation; she could not run small experiments without committing to the full maneuver. So her adaptive law sensibly refused to update its parameter estimates beyond what her prior already supplied. She stayed on the prior. This is what robust adaptive laws do formally: when persistency of excitation is absent, the law stays on its prior rather than drifting in the unexcited directions. The kid did this without naming it; this chapter writes down the formal version.

§ 13.1

# *The Curvature Floor, with adaptive terms reinstated*

Recall the full inequality, which the framework states as a conjecture on the right-hand side and a meaningful statement only after $S(j\omega)$ is defined for the adaptive closed loop:

$$\int_0^\infty \log|S(j\omega)|\,d\omega \;\geq\; \pi\sum_i \mathrm{Re}(p_i^+) \;+\; \alpha\,\mathrm{tr}(\boldsymbol{\Gamma}^{-1}) \;+\; \beta\,\mathrm{curv}(\bar{\Theta}).$$

**What $S(j\omega)$ means for an adaptive closed loop.** The sensitivity function is a standard LTI object. To give the integral a meaning under adaptation, the framework adopts one of the three standard interpretations from the adaptive-control literature:

1. *Frozen-time linearization.* At each time $t$, freeze the parameter estimate $\hat{\boldsymbol{\theta}}(t)$ and compute $S(j\omega;t)$ as the sensitivity of the corresponding LTI closed loop. The Curvature Floor inequality is then claimed for the time-average $\overline{\log|S(j\omega;t)|}$.

2. *Empirical frequency response.* Estimate $|S(j\omega)|$ from input-output data via spectral methods (Welch periodograms, subspace identification). The integral is then over the measured spectrum.

3. *Local LTV operator norm.* For a slowly-varying linear time-varying closed loop, $|S(j\omega)|$ is replaced by the frozen-time operator's $L_2$-gain at frequency $\omega$.

The ICARUS empirical measurement reported in §13.3 uses interpretation (2) on flight-test data. Interpretations (1) and (3) are the analytical settings under which the conjecture would be stated; we do not claim a proof in any of them. The conjecture's

open status (§13.4) is the open status of *all three* interpretations of the inequality. Three terms on the right; three sources of sensitivity-function peaking that the closed-loop must pay for. Let me walk them.

**Term 1: $\pi \sum_i \mathrm{Re}(p_i^+)$.** The Bode-classical right-half-plane contribution. Chapter 7 §7.3 developed this in full. The integrated $\log|S|$ has to be at least this big because right-half-plane open-loop poles force peaks in $|S|$ somewhere in the frequency band; you cannot escape them. For an open-loop stable plant, this term is zero (no $p_i^+$ to sum). For an unstable plant, it is positive.

**Term 2: $\alpha \,\mathrm{tr}(\mathbf{\Gamma}^{-1})$.** The cost of having adaptation at all. The matrix $\mathbf{\Gamma}$ is the adaptive law's gain matrix from Chapter 12; its inverse $\mathbf{\Gamma}^{-1}$ is the Riemannian metric on the parameter manifold $\Theta$. The trace of that metric measures, dimensionlessly, how much "parameter inertia" the adaptive law has—how much the parameter estimate resists being moved by a unit-norm tracking error. Slow adaptation (small $\mathbf{\Gamma}$, large $\mathbf{\Gamma}^{-1}$) has high trace; fast adaptation (large $\mathbf{\Gamma}$, small $\mathbf{\Gamma}^{-1}$) has low trace.

The framework's claim, which is the conjecture's content, is that running with any nonzero adaptation gain contributes a strictly positive amount to the integrated $\log|S|$ floor. *Adaptation is not free.* Even with no right-half-plane poles, an adaptive controller has a Bode-integral floor strictly greater than zero, and the floor scales with $\mathrm{tr}(\mathbf{\Gamma}^{-1})$. The constant $\alpha$ is the conversion factor between parameter-manifold inertia and frequency-domain budget. The framework conjectures that $\alpha > 0$ for any nontrivial adaptive law; its precise value depends on the closed-loop's response characteristics and on the adaptive law's specific form.

**Term 3: $\beta \,\mathrm{curv}(\bar{\Theta})$.** The cost of a curved parameter manifold. For constant $\mathbf{\Gamma}$, the parameter manifold $\Theta = \mathbb{R}^d$ in the $\mathbf{\Gamma}^{-1}$ metric is flat—the scalar curvature is zero, and this term vanishes. For state-dependent or time-varying $\mathbf{\Gamma}$, which arises in recursive-

least-squares adaptive laws and in modulated MRAC variants, the manifold has nonzero curvature, and this term contributes positively to the integrated $\log|S|$ budget.

The scalar curvature $\mathrm{curv}(\bar{\Theta})$ here is the framework's design-choice version of curvature on the parameter manifold (the same kind of distinction we made in Chapter 10 §10 for the Kalman case): a non-negative dimensionless quantity, motivated by Riemannian-geometric intuition but defined as a design choice rather than derived as the literal Ricci scalar of the $\mathbf{\Gamma}^{-1}$ metric. The constant $\beta$ converts this scalar curvature into frequency-domain budget.

**What the inequality reads as.** Adding up the three terms: the integrated $\log|S|$ floor for an adaptive closed-loop is the classical Bode floor (for unstable plants) plus the adaptation-inertia tax plus the parameter-manifold-curvature tax. The three taxes are additive in the framework's conjectural form. Each is non-negative; each can be zero (no right-half-plane poles, no adaptation, constant adaptation gain); when they are all zero, the inequality is equality and reduces to the classical Bode result.

This is the framework's structural claim about the cost of adaptive control. The next sections develop what robust adaptive laws do to the trace and curvature terms, and what the ICARUS implementation measures empirically.

§ 13.2

## *Sigma-modification: bounding parameter drift*

The simplest practical robust adaptive law is sigma-modification (Ioannou and Kokotović, 1983). Where the basic MRAC adaptive law of Chapter 12 was

$$\dot{\hat{\boldsymbol{\theta}}} \;=\; \boldsymbol{\Gamma}\,\boldsymbol{\phi}(\mathbf{x})\,\mathbf{B}^{\mathrm{T}}\mathbf{P}\,\mathbf{e},$$

the sigma-modified version is

$$\dot{\hat{\boldsymbol{\theta}}} \;=\; \boldsymbol{\Gamma}\,\boldsymbol{\phi}(\mathbf{x})\,\mathbf{B}^{\mathrm{T}}\mathbf{P}\,\mathbf{e} \;-\; \sigma\,\hat{\boldsymbol{\theta}},$$

with $\sigma > 0$ a small positive constant. The added term $-\sigma\hat{\boldsymbol{\theta}}$ biases the parameter estimate toward zero in the absence of an excitation signal. When persistency of excitation is present, the regressor-driven term dominates and the estimate tracks the true parameter within a small biased ultimate bound (the $\sigma$ term creates a residual bias toward zero that scales with $\sigma/\lambda_{\min}$(PE covariance)); when PE is absent, the regressor-driven term goes to zero (no tracking error to drive it) and the sigma term pulls $\hat{\boldsymbol{\theta}}$ back toward the origin at exponential rate. Exact convergence is given up in exchange for the boundedness guarantee.

**Why sigma-modification works.** Without modification, the basic MRAC adaptive law can suffer *parameter drift*: during extended periods of no PE, small disturbances accumulate in the parameter estimate, and the estimate slowly wanders away from the truth. When PE eventually returns, the closed-loop sees a transient as the estimate re-converges. In flight-test history, this is the recurring failure mode of adaptive controllers.

Sigma-modification fixes this by making the parameter es-

timate equation a damped one. Even if no regressor signal is present, $\hat{\boldsymbol{\theta}}$ exponentially decays toward zero at rate $\sigma$. The estimate cannot drift unboundedly. When regressor signal returns, the estimate resumes converging to the true parameter, with a transient that is now bounded by how far $\hat{\boldsymbol{\theta}}$ had decayed during the quiet period.

**The cost of sigma-modification.** The modification is not free. Even at the true parameter $\boldsymbol{\theta}$, the sigma term produces a steady-state bias: in equilibrium, $\boldsymbol{\Gamma}\boldsymbol{\phi}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{e} = \sigma\hat{\boldsymbol{\theta}}$, which means a small persistent tracking error is required to balance the sigma drift. The closed-loop tracking error no longer converges to zero; it converges to a small bounded region whose size depends on $\sigma$ and on the persistent regressor signal.

This is the ultimate-boundedness condition of Chapter 11 §11.2 made operational. Sigma-modification trades exact tracking for guaranteed parameter boundedness. For most flight-control applications, the trade is worth it: a controller that tracks within a small ultimate bound and stays robust to long quiet periods is better than a controller that tracks exactly but drifts during cruise.

**The dead-zone alternative.** An alternative robust modification is the dead-zone:

$$\dot{\hat{\boldsymbol{\theta}}} = \begin{cases} \boldsymbol{\Gamma}\,\boldsymbol{\phi}(\mathbf{x})\,\mathbf{B}^{\mathrm{T}}\mathbf{P}\,\mathbf{e} & \text{if } \|\mathbf{e}\| > \epsilon, \\ \mathbf{0} & \text{if } \|\mathbf{e}\| \le \epsilon, \end{cases}$$

where $\epsilon > 0$ is a small threshold. When the tracking error is below the threshold, the adaptive law is gated off entirely; only above the threshold does it update. This explicitly freezes the parameter estimate during periods of small tracking error, regardless of whether the cause is PE or just good tracking.

Dead-zone modifications are simpler than sigma-modifications and easier to analyze, but they introduce a discontinuity in the

adaptive law that can produce limit cycles in some configurations. In practice, sigma-modification is the more widely used robust variant; the ICARUS implementation uses sigma.

§ 13.3

## *Sidebar: ICARUS empirical measurement of the Curvature Floor*

*The following sidebar is reproduced from the ICARUS architecture documentation, with its honest disclosure about what was and was not quantitatively fitted.*

> ICARUS measures $\int \log|S(j\omega)|\,d\omega$ across $(\mathbf{\Gamma}, \sigma)$ parameter sweeps. The integral grows monotonically with both $\mathrm{tr}(\mathbf{\Gamma}^{-1})$ and modification strength $\sigma$—qualitatively consistent with the conjectured form
>
> $$\int \log|S(j\omega)|\,d\omega \;\geq\; \pi \cdot \sum \mathrm{Re}(p_i^+) + \alpha\,\mathrm{tr}(\mathbf{\Gamma}^{-1}) + \beta\,\mathrm{curv}(\bar{\Theta}).$$
>
> **Quantitative fitting of $\alpha, \beta$ was not attempted** (no closed form for $\mathrm{curv}(\bar{\Theta})$; reliable fit would require $\geq 50$ sweep points; we ran 10). The conjecture remains analytically open.

**Honest unpacking.** Three things to be clear about.

1. *The qualitative monotonicity has been verified.* Running ICARUS's MRAC component across a 10-point sweep

of $(\mathbf{\Gamma}, \sigma)$ values shows that the empirical $\int \log|S|\, d\omega$ grows monotonically with both $\mathrm{tr}(\mathbf{\Gamma}^{-1})$ (slower adaptation $\rightarrow$ larger floor) and with $\sigma$ (more aggressive sigma-modification $\rightarrow$ larger floor). This is the structural prediction the conjectured inequality makes, and the empirical data is consistent with it.

2. *Quantitative fitting was not attempted.* The constants $\alpha$ and $\beta$ have not been numerically determined. To fit them reliably, the framework's documentation reports that $\geq 50$ sweep points would be needed; the deployed work has 10. Beyond that, $\mathrm{curv}(\bar{\Theta})$ has no closed form—the framework defines it as a design choice, which means computing its numerical value requires choosing the specific scalar curvature function, which the deployed work does not formally do.

3. *The conjecture is analytically open.* The inequality has not been proven from first principles. It is a structural hypothesis the framework offers, supported by qualitative empirical monotonicity, that the rigorous closure of (i) deriving the inequality from Lyapunov arguments, (ii) computing $\alpha$ and $\beta$ from the adaptive law's structure, and (iii) verifying the inequality holds across more diverse plant families remains future work.

The framework is allowed to make conjectures. Conjectures are how mathematical research moves forward. What the framework is committed to is naming them as conjectures, not as proven results, and documenting exactly what level of empirical support each conjecture currently has. The Curvature Floor inequality is at the qualitative-empirical-monotonicity stage. It is not at the quantitatively-fitted-constants stage, and it is not at the proven-from-first-principles stage. The deployment of ICARUS

does not depend on resolving the conjecture; ICARUS uses the predicted monotonicity to size its modulation schedules, but does not assume the inequality's exact form.

This is the framework's epistemic stance throughout the book, applied here: name the prediction, name what verifies it, name what does not verify it, do not claim more than the evidence supports.

§ 13.4

## *Why this conjecture matters operationally*

Suppose, hypothetically, that the inequality is true with computable $\alpha$ and $\beta$. What would change?

**Adaptive-controller certification could lean on the inequality.** The aerospace certification process for adaptive flight controllers is famously hard. Regulators want analytical bounds on the closed-loop's worst-case behavior, and adaptive systems have historically resisted such bounds because the parameter estimate's trajectory depends on the realized signal. The Curvature Floor inequality, if proven, would give a per-design budget on the integrated $\log|S|$ that the adaptive law contributes. A designer could choose $\mathbf{\Gamma}$ and the sigma-modification strength $\sigma$ to achieve a specified target on the integrated floor; certification could verify that target against the proven bound. This would be a structural certification gain.

**The $(\mathbf{\Gamma}, \sigma)$ design problem would become geometric.** Currently, adaptive-controller designers tune $\mathbf{\Gamma}$ and $\sigma$ heuristically, by running simulations and adjusting until the closed-loop behaves well. The inequality would convert this into a constrained-optimization problem: pick $(\mathbf{\Gamma}, \sigma)$ that minimizes adaptive-

controller bandwidth loss subject to the integrated-floor budget. The geometric content of the problem—the curvature of the parameter manifold under the chosen $\mathbf{\Gamma}^{-1}$ metric—would become a first-class object in the design process.

**The framework's robustness story would close.** Chapters 7 through 12 have built the Curvature Floor's three terms one at a time. The proof would close the book's epistemic loop: the framework would have a quantitative bound on the cost of adaptation, derivable from first principles, verifiable across deployments. That would let the framework's universality claim extend from "works empirically on this benchmark" to "has a quantitative theory behind why it works."

The framework does not depend on this closure. ICARUS works without the inequality being proven; the modulation schedules are tuned empirically, and the verified benchmark results of Chapter 20 stand on their own. But the conjecture is one of the cleanest open problems the framework has surfaced, and resolving it would be the kind of result that turns a working architecture into a working theory.

§ 13.5

## *What you take from this chapter*

Four things.

**First.** The Curvature Floor inequality

$$\int_0^\infty \log|S(j\omega)|\,d\omega \;\geq\; \pi \sum_i \mathrm{Re}(p_i^+) \;+\; \alpha\,\mathrm{tr}(\mathbf{\Gamma}^{-1}) \;+\; \beta\,\mathrm{curv}(\bar{\Theta})$$

has three contributions: the classical Bode term (right-half-plane

open-loop poles), the adaptation-inertia term ($\mathbf{\Gamma}^{-1}$ trace), and the parameter-manifold-curvature term ($\beta$ scalar curvature). The framework conjectures that all three are present and additive for adaptive closed-loops.

**Second.** Robust adaptive laws—sigma-modification and dead-zone modifications—bound parameter drift during periods without persistency of excitation. They trade exact tracking for ultimate boundedness. Sigma-modification is the standard practical choice; the ICARUS architecture uses it.

**Third.** The ICARUS architecture has measured the qualitative monotonicity predicted by the Curvature Floor inequality across 10-point sweeps of $(\mathbf{\Gamma}, \sigma)$. The quantitative fitting of $\alpha, \beta$ has not been attempted. The conjecture remains analytically open.

**Fourth.** The framework names the conjecture as a conjecture, not as a proven result. The deployment of ICARUS does not depend on resolving it. Closing the conjecture is one of the cleanest open problems the framework has surfaced, and resolution would convert the framework's robustness story from empirical to theoretical.

This closes Part IV. Part V opens with Chapter 14: inertial mechanization on SO(3) and SE(3), the quaternion strapdown algorithm, and the sensor-fusion bundle that ICARUS reads via GQL.

*Vamos.*

Chapter 14

# *Inertial mechanization and sensor fusion*

*The problem of navigation is the problem of knowing where you are when no one is there to tell you.*
*Charles Stark Draper, MIT Instrumentation Laboratory (paraphrased from the Apollo navigation lectures, c. 1965)*

## § 14.0 *What inertial navigation is doing*

Open Part V. The previous parts have been about control: given the state, what command should the controller produce? This part is about *knowing the state in the first place*—navigation, sensor fusion, the question of where the vehicle actually is.

Inertial navigation is the oldest answer to that question. The idea, in plain language: if you know where you started, and you have a sensor that measures every change in motion the vehicle experiences, you can integrate those changes forward in time to

keep a running estimate of where the vehicle is now. No external reference needed. No radio signal. No star fix. Just the sensor on board and the math.

**The bike analogy.** Go back to the kid in §1.0 one more time. Imagine the kid rode a bicycle through a tunnel where she could not see anything outside. She knows where she started: at the tunnel entrance. She can feel every pedal stroke, every turn of the handlebars, every bump in the road. From those felt sensations, she could in principle reconstruct her current position inside the tunnel: how far she has gone, which direction she is facing, whether she has climbed or descended. She does not need to see anything; she needs to keep a running integral of her own sense of motion.

That is what inertial navigation does. The vehicle has sensors—accelerometers and gyroscopes, packaged into a unit called an *IMU* (Inertial Measurement Unit)—that measure the linear acceleration and angular velocity of the vehicle in the vehicle's own body frame. The navigation algorithm integrates those measurements forward in time, propagating an estimate of position, velocity, and attitude that starts from the known initial state and is refined by every new measurement.

The math is older than aerospace. Newton's second law and the Euler rigid-body equations are eighteenth-century mathematics. What is twentieth-century is the engineering: building accelerometers and gyroscopes precise enough to integrate over hours of flight without losing the position. Charles Stark Draper's MIT Instrumentation Laboratory built the first practical inertial navigators in the 1950s; the Apollo guidance computers ran inertial-only navigation for hours of translunar coast, supplemented by occasional star fixes. The same machinery, with modern silicon, runs on every commercial aircraft, every cruise missile, every spacecraft, and every smartphone.

This chapter develops the math. It runs through what an IMU measures, the strapdown integration algorithms on SO(3) and SE(3) that propagate attitude and pose, where the integration error comes from, and the framework's reading of the inertial sensor as a fiber bundle. The Kalman filter from Chapter 10 returns here in its standard sensor-fusion role: blend the inertial dead-reckoning with whatever external aiding signals are available (GNSS, vision, magnetometer), with the curvature obstruction $\hat{\kappa}$ gating which aiding source the filter trusts more.

§ 14.1

## *What the IMU's sensors measure*

An inertial measurement unit has two kinds of sensors: *accelerometers* and *gyroscopes* (gyros for short). They are mounted rigidly in the vehicle's body, in three orthogonal axes each. Together they give six measurements per sample, taken at high rate (typically 200 Hz to 1 kHz on modern hardware).

**The accelerometer.** Each accelerometer measures the *specific force* along one body axis. Specific force is not quite the same as acceleration; it is the acceleration relative to free-fall:

$$\mathbf{f}_{\text{spec}} \;=\; \mathbf{a}_{\text{inertial}} \;-\; \mathbf{g}.$$

Here $\mathbf{a}_{\text{inertial}}$ is the actual inertial acceleration of the vehicle, and $\mathbf{g}$ is the local gravity vector. A stationary accelerometer on a table reads $+\mathbf{g}$ (because the table pushes up, producing specific force pointing away from Earth's center). A free-falling accelerometer reads zero. The distinction matters: the accelerometer is not measuring acceleration directly; it is measuring the force the structure has to apply to keep its test mass from falling freely.

For navigation, you need true inertial acceleration. So the navigation algorithm has to add back the gravity vector:

$$\mathbf{a}_{\text{inertial}} = \mathbf{f}_{\text{spec}} + \mathbf{g}.$$

This requires knowing where you are (because gravity varies with latitude and altitude). The gravity correction is the first thing the navigation algorithm computes, and it is one of the reasons inertial navigation cannot be done in isolation from a position estimate.

**The gyroscope.** Each gyro measures the *angular velocity* of the body frame about one body axis, relative to inertial space. The output is $\boldsymbol{\omega}_{\text{body}}$, a three-component vector that says how fast the vehicle is rotating, expressed in the body's own frame. Unlike the accelerometer, the gyro's measurement is direct: gyros sense rotation, and angular velocity is what the navigation algorithm needs.

Modern gyros come in several flavors: ring laser gyros (RLGs), fiber optic gyros (FOGs), and microelectromechanical (MEMS) gyros. They differ in cost, weight, noise, and drift. A high-grade RLG might drift $0.001^\circ$/hour; a MEMS gyro might drift $1^\circ$/hour. The grade of gyro you use determines how often you have to correct the navigation solution with an external fix; that calculation is in §14.5.

**What the IMU does not give you directly.** The IMU outputs specific force and angular velocity. It does not output position, velocity, or attitude. The navigation algorithm has to integrate the IMU's outputs forward in time, starting from a known initial state, to produce the running estimate of where the vehicle is and how it is oriented. That integration is what the next two sections develop.

§ 14.2

## *Strapdown integration on* SO(3)*: the attitude propagation*

The vehicle's attitude—its orientation relative to a reference frame—is an element of SO(3) at every instant. As Chapter 3 §3.4 developed, SO(3) is non-trivially topological, with $\pi_1(\mathrm{SO}(3)) \cong \mathbb{Z}/2$, and the safe representation for navigation is the unit quaternion $q \in S^3$, the universal cover of SO(3).

Strapdown attitude propagation uses the quaternion kinematic equation from Chapter 5 §5.2:

$$\dot{q}(t) \;=\; \tfrac{1}{2}\, q(t) \otimes \boldsymbol{\omega}_{\mathrm{pure}}(t).$$

Here $\boldsymbol{\omega}_{\mathrm{pure}} = 0 + \omega_1 i + \omega_2 j + \omega_3 k$ is the body-frame angular velocity (from the gyro), treated as a pure imaginary quaternion, and $\otimes$ is quaternion multiplication. The equation says: the attitude's time derivative is half the attitude multiplied by the angular velocity.

**The discrete-time strapdown algorithm.** At sample rate $\Delta t$ (typically 1 ms for a 1-kHz IMU), the navigation algorithm reads $\boldsymbol{\omega}_k$ from the gyro and propagates the attitude:

$$q_{k+1} \;=\; q_k \otimes \exp\!\big(\tfrac{1}{2}\,\boldsymbol{\omega}_k\,\Delta t\big),$$

where exp is the quaternion exponential of a pure imaginary quaternion (a closed-form formula involving sines and cosines of the magnitude). This is the *strapdown algorithm*: integrate the gyro forward, one sample at a time, by multiplying the previous attitude by the small-step rotation the gyro just measured.

Several technical conditions: the closed-form exponential preserves the unit-norm constraint $|q| = 1$ exactly, so the propagated

quaternion stays on $S^3$. The algorithm renormalizes $q$ every few hundred samples anyway, to absorb floating-point drift. The propagation itself runs without any sign correction—enforcing $q_0 \geq 0$ on the propagated state would introduce an artificial discontinuity at $\theta = \pi$ exactly where the maneuver is most stressed. Continuity is preserved by letting the propagated quaternion follow whichever lift its initial condition started on. The short-arc convention is applied separately, at the consumer end: controllers compute their attitude error $q_{\text{err}} = q_{\text{cmd}}^{-1} \otimes q_{\text{actual}}$ and choose the sign of $q_{\text{err}}$ that minimizes the rotation magnitude. This keeps the propagated state continuous and the error signal free of spurious sign flips, both at the same time.

**What can go wrong.** The strapdown algorithm is exact in continuous time but quantized in discrete time. Each $\Delta t$ step introduces a small error from the finite-step approximation; over thousands of samples, those errors accumulate. More importantly, the gyro has its own errors: bias (a constant offset added to every measurement), scale-factor error (the gyro reports a multiple of the true rotation), random noise, and temperature drift. Each of these contaminates the propagated attitude.

Without external aiding, attitude error grows roughly linearly in time at a rate set by the gyro's bias. A 1°/hour gyro drifts 1° of attitude in an hour. For an aircraft in a one-hour flight, that is acceptable for some purposes (the aircraft is still upright) and unacceptable for others (a 1° attitude error at 5000 ft altitude translates to several hundred feet of horizontal position error after a few minutes of integration). External aiding—GNSS for position, magnetometer for heading, vision for relative motion—is what keeps the propagated attitude bounded over long flights.

§ 14.3

## *Strapdown integration on* SE(3)*: full pose*

Attitude alone is not enough; the navigation algorithm also has to track position and velocity. Together, attitude and position form the vehicle's *pose*, an element of the special Euclidean group $\mathrm{SE}(3) = \mathrm{SO}(3) \ltimes \mathbb{R}^3$ (a rotation paired with a translation, with the semi-direct product structure that makes the group non-abelian).

**The velocity update.** Velocity propagation uses the accelerometer:

$$\dot{\mathbf{v}}_{\mathrm{nav}}(t) \;=\; \mathbf{R}(q(t))\,\mathbf{f}_{\mathrm{spec}}(t) \;+\; \mathbf{g}_{\mathrm{nav}},$$

where $\mathbf{R}(q)$ is the body-to-navigation-frame rotation matrix (computed from the propagated quaternion), $\mathbf{f}_{\mathrm{spec}}$ is the specific force in body frame (from the accelerometer), and $\mathbf{g}_{\mathrm{nav}}$ is the gravity vector in the navigation frame. The body-frame specific force is rotated into the navigation frame using the current attitude, and gravity is added back to recover the true inertial acceleration.

**The position update.** Position then integrates the velocity:

$$\dot{\mathbf{r}}_{\mathrm{nav}}(t) \;=\; \mathbf{v}_{\mathrm{nav}}(t).$$

This is the trivial integration. The non-trivial part is making sure the velocity is right, which depends on the attitude being right, which depends on the gyro being good. Every error propagates: gyro error $\to$ attitude error $\to$ wrong rotation of specific force $\to$ wrong velocity $\to$ wrong position.

**The full strapdown algorithm, in one sentence.** Each sample, the algorithm reads gyro and accelerometer, propagates

the quaternion attitude one step with the gyro, rotates the accelerometer's specific force into navigation frame using the propagated attitude, adds gravity, integrates the result into velocity, integrates velocity into position. Ten or more state components (the four-component quaternion attitude or its three-parameter local error, the three velocity components, the three position components, and—in any production filter—the three accelerometer biases and three gyro biases), updated every $\Delta t$, running on whatever processor the avionics has. The pose itself exposes six numbers to downstream consumers (three attitude, three position), but the mechanization state the algorithm carries internally is larger. The Apollo guidance computer ran this algorithm at 25 Hz with 4 KB of RAM. Modern systems run it at 1 kHz with megabytes of state to spare.

§ 14.4

## *Why the integration drifts*

Pure inertial navigation is dead-reckoning: every error in the sensor's measurement accumulates over time in the propagated state. There are three categories of error worth naming.

**Bias.** Each accelerometer and gyro has a small constant offset added to every measurement—typically a few milli-$g$ on the accelerometer, a few degrees-per-hour on the gyro. The bias is mostly constant over an hour-long flight but slowly drifts with temperature and time. Bias accumulates in the integration: an accelerometer bias of $0.001\,g$ produces a velocity error growing at $0.01\,\mathrm{m/s}$ per second, and a position error growing quadratically at $0.005\,\mathrm{m/s^2}$ per second per second. After an hour, that single milli-$g$ of accelerometer bias produces about 65 km of position

error.

**Scale factor and misalignment.** The sensor's reported value is not exactly the true value; it is $(1 + \epsilon)$ times the true value for some small $\epsilon$, plus a small rotational misalignment that couples one axis's input into another axis's output. These errors are usually calibrated out at the factory, but residual scale factor and misalignment of a few hundred parts per million remain.

**Random noise.** Each measurement has random noise on top of the systematic errors. Noise integrates as a random walk: position error grows like $t^{3/2}$ from accelerometer noise, attitude error grows like $t^{1/2}$ from gyro noise. Random walk grows slower than bias-driven errors over short times, but over very long times it eventually dominates.

**What this means operationally.** Pure inertial navigation is excellent over short timescales (seconds to minutes) and unreliable over long ones (hours to days, depending on grade). Every operational system pairs inertial navigation with at least one external aiding source: GNSS for position (most common), barometric altimeter for vertical, magnetometer for heading, star tracker for spacecraft, visual odometry for ground vehicles. The aiding source is intermittent (a star fix every few minutes; a GNSS fix every second when sky-visible); inertial fills in between. The Kalman filter from Chapter 10 is what optimally blends them.

§ 14.5

## *Sensor fusion: blending inertial with external aiding*

The standard architecture is the *loosely coupled* or *tightly coupled* INS/GNSS Kalman filter. Both architectures use the inertial system's propagated state as the primary estimate, and use GNSS (or other aiding) measurements to correct the inertial estimate at the aiding measurement's rate.

**The Kalman framing.** The state vector includes position, velocity, attitude (as quaternion or rotation vector), and the sensor errors (accelerometer biases, gyro biases). The propagation step uses the IMU at high rate (1 kHz). The update step uses the aiding sensor at lower rate (1–10 Hz for GNSS). Each update step uses the difference between the predicted aiding measurement (from the current state estimate) and the actual measurement to correct the state—exactly the structure Chapter 10 §10.1 developed.

The bias states are crucial: by including the accelerometer and gyro biases in the filter's state vector, the filter can estimate the biases online from the aiding signal, and use the estimated biases to compensate the raw IMU outputs. This is how a tactical-grade IMU paired with GNSS aiding can achieve performance close to a navigation-grade IMU running standalone: the GNSS observations let the filter calibrate the IMU's biases continuously.

**The framework's reading.** In the bundle language of Chapter 5, the sensor-fusion architecture is a single bundle whose base is time and whose fiber is the joint state (position, velocity, attitude, biases). The IMU drives the section's smooth evolution (the prediction step); the aiding signal injects information at discrete events (the update step). The bundle's curvature

obstruction $\hat{\kappa}$, as developed in Chapter 10 §10, measures how confident the filter is in its current state estimate; ICARUS gates downstream controller decisions on this $\hat{\kappa}$.

**ICARUS sensor-fusion sidebar (already given in Ch.10, §10.5).** The verbatim ICARUS sidebar from Chapter 10 §10.5 covers the architectural commitment: sensor-fusion residuals computed by the sensor thread, stored on the `sensor_fusion` bundle, read by the flight loop via `SECTION sensor_fusion AT t = @latest_t`, with $\hat{\kappa}$ read from `CURVATURE sensor_fusion`. Strapdown integration runs on the sensor thread; the flight loop reads the fused state through GIGI; the controller decisions of Chapters 8 and 9 use the read state directly.

§ 14.6

## *The inertial bundle, geometrically*

The strapdown algorithm has a clean fiber-bundle reading worth naming, because it ties Chapter 3's formal vocabulary to the runtime navigation pipeline.

**The base manifold.** The base of the inertial bundle is time, $[0, T] \subset \mathbb{R}$. The navigation algorithm propagates the state forward along this base.

**The fiber.** Over each instant of time, the fiber is the pose space $\mathrm{SE}(3) \times \mathbb{R}^3 \times \mathbb{R}^6$, encoding attitude (in SO(3), lifted to $S^3$), position (in $\mathbb{R}^3$), velocity (in $\mathbb{R}^3$), and sensor biases (six components: three accelerometer, three gyro). The dimension of the fiber is $3 + 3 + 3 + 6 = 15$ at minimum; more elaborate filters add additional bias states for scale factor, alignment, and lever arm.

**The section.** The navigation state at time $t$ is a section of the bundle—a smooth assignment of pose, velocity, and biases to each time. The strapdown algorithm propagates the section forward; the aiding sensor corrects it.

**The connection 1-form.** The IMU's instantaneous output—specific force in body frame and angular velocity in body frame—is, structurally, the connection 1-form of the inertial bundle, evaluated on the time-axis tangent vector. The strapdown algorithm is the parallel-transport equation on this bundle, integrated along time. This is exactly the structure of Chapter 3 §3.2's parallel transport, specialized to the inertial bundle.

**The curvature.** The closed-loop coupling map's curvature here is the rate at which sensor errors propagate into position error. Highly observable maneuvers (high-acceleration turns, attitude changes) make the coupling well-conditioned (low curvature, high observability of biases); low-acceleration straight-and-level flight makes the coupling poorly-conditioned (high curvature, biases harder to estimate). The framework's $\hat{\kappa}$ for the sensor-fusion bundle measures this conditioning, and ICARUS gates the controller on it.

## § 14.7

## *What you take from this chapter*

Four things.

**First.** An IMU measures specific force and angular velocity in the vehicle's body frame. The navigation algorithm propagates attitude (via the quaternion strapdown equation), velocity (via gravity-corrected specific-force integration), and position (via

velocity integration) starting from a known initial state. The integration runs at high rate (often 1 kHz).

**Second.** Pure inertial navigation drifts. Bias, scale-factor, misalignment, and noise all accumulate; without external aiding, position error grows from milliseconds-of-bias into kilometers-of-error over an hour. Practical inertial-navigation systems always pair the IMU with at least one external aiding source.

**Third.** The standard sensor-fusion architecture is the INS/GNSS Kalman filter: IMU propagation at high rate, GNSS (or other aiding) updates at low rate, joint state including sensor biases that get estimated online from the aiding signal. The Kalman filter of Chapter 10 is what runs the blend.

**Fourth.** In the framework's bundle language, the inertial system is a fiber bundle whose base is time, whose fiber is the joint pose-velocity-bias state, and whose connection 1-form is the IMU's instantaneous output. The strapdown algorithm is parallel transport on this bundle. The framework's $\hat{\kappa}$ measures the conditioning of the bias-observability problem, and ICARUS reads $\hat{\kappa}$ to gate downstream controller decisions through GIGI's substrate.

Chapter 15 (next) develops the GNSS/INS integration in more detail—multi-sensor fusion as a coproduct in the category of fiber bundles, Fisher-manifold reading of multi-sensor information. Chapter 16 extends to simultaneous localization and mapping: state-and-map estimation as a joint section over a coupled base. The Kalman machinery of Chapter 10 runs through all of it.

*Vamos.*

Chapter 15

# *GNSS/INS integration as bundle composition*

*No single navigation sensor is complete. Each has errors the others can correct, and integration is the art of letting them.*
*Paul D. Groves, Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems (2nd edition, Artech House, 2013), paraphrased from the introduction*

## § 15.0 *The bike comes out of the tunnel*

Chapter 14 left the kid in a tunnel, dead-reckoning from felt sensations. She knew where she had started, she could feel every pedal stroke and handlebar turn, and from those felt sensations she was integrating forward a running estimate of where she was inside the tunnel. The integration was good for a few hundred meters, then started drifting; over a long enough tunnel the estimate would wander far enough off that she would not know

where the exit was.

Now imagine she comes out of the tunnel.

The sky is open. There are landmarks. There is a phone in her pocket that, every few seconds, tells her where she is to within ten meters via signals from satellites overhead. The phone's fixes are not perfect—each one has some noise, the satellites are not always all visible, when the kid rides past tall buildings the signal weakens or drops out entirely—but the fixes do not drift over time. Whatever inaccuracy a single fix has is the same inaccuracy the next fix has; the errors do not accumulate. The dead reckoning from the pedals is the opposite: it is smooth and immediate, but it slowly walks away from the truth.

The kid now has two information sources with complementary error structures. The pedals give a continuous, low-noise, but drifting position estimate. The phone gives an intermittent, noisier, but bounded position estimate. The right answer is to combine them: trust the phone for the long-term anchor, trust the pedals for the smooth interpolation between phone fixes. Neither alone is sufficient; together they keep the estimate tight and stable for an arbitrarily long ride.

This chapter is the formal version of that. Inertial navigation (the pedals) and Global Navigation Satellite Systems (the phone) have complementary error characteristics by design, and modern navigation systems blend them through a Kalman filter (Chapter 10) over a single fused state. The framework's reading of this blending is structural: GNSS/INS integration is a *coproduct* in the category of fiber bundles. Two sensors, each a bundle of measurements over time; one fused bundle whose fiber carries the information of both. The Kalman filter is the structure that operates on sections of the coproduct.

The chapter covers six things:

1. Why GNSS and INS have complementary errors, and what

that means operationally. (§15.1)

2. The standard integration architectures—loosely coupled, tightly coupled, deeply coupled. (§15.2)
3. Multi-sensor fusion as a coproduct in the category of fiber bundles. (§15.3)
4. The Fisher-information reading of multi-sensor measurement combination. (§15.4)
5. How ICARUS stores and queries the multi-sensor fusion bundle through GIGI. (§15.5)
6. Forward pointer to SLAM in Chapter 16, where the map itself joins the state. (§15.6)

§ 15.1

## *Complementary errors: why GNSS and INS need each other*

Before any equations, here is the physical reason these two sensors are paired in every modern navigation system on the planet.

**What INS gives you.** An inertial navigation system (Chapter 14) integrates accelerometer and gyro measurements forward in time. At high rate (typically 1 kHz), it produces a smooth, low-latency estimate of position, velocity, and attitude. Between measurements there is nothing to wait for—the next position estimate is one millisecond from the last, and the next attitude estimate is one millisecond from the last. For control loops

running at 200–400 Hz, this is the right kind of signal: dense, continuous, available instantly.

What INS gives you operationally: *smoothness*, *low noise over short windows*, and *no dependence on anything external.*

What INS does not give you: a stable long-term position. Bias and noise in the sensors integrate over time. A $0.001\,g$ accelerometer bias produces about 65 km of position error after an hour (Chapter 14 §14.4). Pure INS is excellent for seconds-to-minutes timescales and unreliable for hours-to-days.

**What GNSS gives you.** A Global Navigation Satellite System receiver—GPS, GLONASS, Galileo, BeiDou, or any combination—measures the ranges from the receiver to several satellites whose positions are known. From four or more such ranges, the receiver computes its own position (three coordinates) plus the receiver clock's offset from the satellite-system time (one variable). At consumer-grade single-frequency receivers, the position fix is accurate to about 5–10 meters, 95% of the time, under open-sky conditions. At survey-grade dual-frequency carrier-phase receivers, it is accurate to a few centimeters.

The fix arrives once or twice per second on consumer hardware, faster on specialized units. It is intermittent (urban canyons, tunnels, indoors block the signal). It has its own noise structure: pseudorange measurements have multipath effects, ionospheric delays, satellite clock drift. But the noise is *bounded*: GNSS errors do not accumulate over time the way INS errors do. A receiver that has been stationary for an hour gives the same accuracy as a receiver that has been stationary for a minute. The satellites do not drift.

What GNSS gives you operationally: *long-term boundedness*, *absolute position* (anchored to Earth-fixed coordinates), and *no integration of small errors.*

What GNSS does not give you: a continuous high-rate stream. The fixes are slow, intermittent, sometimes absent for tens of

seconds. The receiver does not directly measure attitude (except indirectly through the velocity vector at high speeds). And GNSS depends on signals from above; under any obstruction, it goes blind.

**Why the pair is perfect.** INS errors grow over time but the sensor is fast and dense. GNSS errors are bounded over time but the sensor is slow and intermittent. The Kalman filter from Chapter 10 is what blends them: at every GNSS fix, the filter corrects the INS state using the new external information; between fixes, the INS propagates the state smoothly. The propagated state inherits the GNSS's long-term boundedness through the corrections, and inherits the INS's high-rate smoothness through the propagation.

This pairing is not a clever engineering hack. It is the only architecture that gives both properties at once. Every modern navigation system—commercial aircraft, missiles, spacecraft, automobiles, smartphones—uses some variant of it.

§ 15.2

## *Integration architectures: loose, tight, deep*

There are three standard architectures for INS/GNSS integration, named for how directly the GNSS information enters the Kalman filter.

**Loosely coupled.** The GNSS receiver runs as a standalone unit. It does its own position-velocity-time computation from satellite measurements and outputs a position fix to the INS. The INS treats this fix as a measurement and runs a Kalman update against it: *the predicted position from INS integration*

versus *the GNSS-reported position*, with the difference driving the Kalman gain to correct the INS state and the sensor biases.

This is the simplest architecture, the most widely deployed in commercial aircraft, and the easiest to certify. The two systems are modular: a GNSS receiver from one vendor can be paired with an INS from another, with the integration logic on the avionics platform.

What it loses: when the GNSS receiver is tracking only three satellites (not enough for a 4D fix), the receiver outputs nothing, and the INS gets no aiding. Even partial GNSS information is unavailable to the navigation filter.

**Tightly coupled.** The GNSS receiver outputs its raw *pseudoranges*—one per visible satellite—to the integration Kalman filter, which uses them directly. The filter's state includes position, velocity, attitude, sensor biases, and the receiver clock offset, all jointly estimated.

This architecture handles partial GNSS information gracefully: with three satellites, the filter gets three pseudoranges' worth of correction (less than the four needed for a standalone fix, but still informative); with two satellites, it gets two pseudoranges; with one, one. The integration extracts the maximum information from whatever satellites are currently visible.

The cost is architectural complexity: the GNSS receiver must expose raw pseudoranges, the filter must model receiver clock dynamics, and the certification path is longer.

**Deeply coupled (or ultra-tight coupling).** The INS state feeds back into the GNSS receiver's signal-tracking loops. A receiver with an aiding inertial signal can track satellites at lower carrier-to-noise ratios than a standalone receiver, because it knows where the signal should be (from the inertial-aided Doppler estimate) and can use a narrower tracking bandwidth.

This is the architecture used in modern military and high-

precision civilian systems. The deeply coupled receiver maintains lock on satellites that a standalone receiver would lose, at the cost of even tighter integration between the receiver firmware and the navigation filter. Certification is harder still.

**What ICARUS does.** The ICARUS architecture is tightly coupled. The sensor thread (Chapter 10 §10.5) maintains the joint state—position, velocity, attitude, IMU biases, GNSS clock offset—and processes pseudoranges directly as measurements. The flight loop reads the fused state through GIGI. The substrate's curvature obstruction $\hat{\kappa}$ for the sensor-fusion bundle takes into account both the IMU's quality and the current satellite geometry.

§ 15.3

## *Multi-sensor fusion as a coproduct*

Now the framework's structural reading. This is where Chapter 3's fiber-bundle vocabulary earns its keep.

**The two sensor bundles.** Each sensor produces a stream of measurements over time. Treat each stream as a section of a fiber bundle:

- *The INS bundle* $E_{\text{ins}}$. Base: time $[0, T]$. Fiber over each $t$: the inertial state space (pose, velocity, biases). A section is the inertial-only navigation solution.
- *The GNSS bundle* $E_{\text{gnss}}$. Base: time $[0, T]$ (with measurements only at the discrete sample times). Fiber over each measurement time: the pseudorange-vector space (one per visible satellite, plus the receiver clock offset). A section is the receiver's measurement stream.

These bundles share a base manifold (time) but have different fibers. They carry information about overlapping but not identical aspects of the world.

**The coproduct.** In the category of fiber bundles over a fixed base, the *coproduct* $E_{\text{ins}} \sqcup E_{\text{gnss}}$ is the bundle whose fiber over each $t$ is the direct sum of the two sensors' fibers:

$$(E_{\text{ins}} \sqcup E_{\text{gnss}})_t \;=\; (E_{\text{ins}})_t \;\oplus\; (E_{\text{gnss}})_t.$$

A section of the coproduct is the simultaneous specification of both sensors' measurements at every time. Categorically, the coproduct satisfies the universal property that any map from *either* sensor's bundle into a third bundle factors uniquely through the coproduct: the coproduct is the smallest bundle that contains both sensors' information without losing any of it.

Operationally: the coproduct bundle is the right state space for the integrated navigation problem. The Kalman filter operates on sections of the coproduct, predicting forward via the INS dynamics (which act on the $E_{\text{ins}}$ summand) and correcting via the GNSS measurements (which constrain the $E_{\text{gnss}}$ summand through the joint state).

**Why "coproduct" and not "product."** The product $E_{\text{ins}} \times E_{\text{gnss}}$ in the category of bundles would also have direct-sum fibers, but it carries a different universal property: maps *into* the product correspond to pairs of independent maps into each factor. The coproduct's universal property—maps *out of* the union correspond to pairs of maps out of each piece—is the right one for sensor fusion. The fused state is the *union of information*, not the product.[1] Either sensor by itself constrains a portion of the state; the fusion is the simultaneous use of both constraints.

[1] In the category of vector bundles over a fixed base, the product and the coproduct coincide as objects: the direct sum $E \oplus F$ is a biproduct, satisfying both universal properties simultaneously. The framework's appeal to the *coproduct's* universal property is therefore not a claim about constructing a

This is the categorical structure that the Kalman filter's covariance update implements. When two sensors observe the same underlying state with independent noise, their information matrices (inverse covariances) add. Information addition is the coproduct structure made operational.

**Generalizing to more sensors.** Adding a magnetometer? Its measurement bundle $E_{\text{mag}}$ joins the coproduct: $E_{\text{ins}} \sqcup E_{\text{gnss}} \sqcup E_{\text{mag}}$. Adding a barometric altimeter? Another summand. Vision odometry? Another summand. The coproduct structure scales without modification, and the Kalman filter's information-addition update scales with it. This is the framework's reason for the architectural commitment: every sensor is a bundle, the navigation state is the coproduct, the filter is the universal-property realization.

§ 15.4

## *The Fisher-information reading*

Chapter 10 §10.3 set up the Fisher information matrix of a Gaussian estimator. In the linear-Gaussian limit, the inverse of the posterior covariance is the Fisher information; the Cramér-Rao bound is saturated by the Kalman filter; the Riemannian metric on the parameter manifold is given by the Fisher information.

For multi-sensor fusion, the Fisher-information reading is particularly clean. If two sensors observe the same state with independent Gaussian noise, their measurement Fisher-information

different bundle, but about which structural reading captures the engineering content of sensor fusion—information addition out of the union, rather than independent observation into the product.

matrices simply add:

$$\mathbf{I}_{\text{fused}}(\boldsymbol{\theta}) \;=\; \mathbf{I}_{\text{ins}}(\boldsymbol{\theta}) \;+\; \mathbf{I}_{\text{gnss}}(\boldsymbol{\theta}).$$

This is the information-addition theorem of statistics, specialized to the linear-Gaussian case with independent measurements of the same state. The posterior covariance is the inverse of the sum:

$$\mathbf{P}_{\text{fused}}^{-1} \;=\; \mathbf{P}_{\text{ins}}^{-1} \;+\; \mathbf{P}_{\text{gnss}}^{-1}.$$

The fused covariance is smaller than either factor's covariance (in the Loewner partial order), and strictly smaller along the directions where both sensors provide information. Directions seen by only one sensor are tightened by that sensor's contribution alone.

**Scope of the formula.** This identity is the *ideal-case* information-filter update: independent Gaussian measurements of the same state, no prior, no process model, no correlated errors, no latency, no bias states, no nonlinear measurement model. Real INS/GNSS fusion has all of those features—priors on the state, a continuous-time process model, correlated errors across measurement epochs, transport delays, ten or more bias states (§14.3), nonlinear pseudorange equations—and the production implementation runs the full extended Kalman or unscented filter rather than the bare information-addition formula. The identity is what the production filter *reduces to* in the linear-Gaussian independent-measurement limit; it is not, by itself, the general fusion law.

**What this means for the framework's design-choice $\hat{\kappa}$.** Following the careful framing of Chapter 10 §10, the framework's curvature obstruction $\hat{\kappa}$ for the sensor-fusion bundle is *not* the Fisher-Ricci scalar of the fused parameter manifold. It is a design-choice scalar, defined as a monotone non-negative function of $\text{tr}(\mathbf{P}_{\text{fused}})$, chosen for engineering convenience and motivated by

the Fisher-geometric intuition.

For multi-sensor fusion, the framework's $\hat{\kappa}$ inherits the additive-information structure: more independent, correctly modeled, informative sensors $\rightarrow$ smaller $\mathrm{tr}(\mathbf{P}_{\mathrm{fused}})$ $\rightarrow$ smaller $\hat{\kappa}$ $\rightarrow$ higher operational confidence $C = 1/(1+\hat{\kappa})$. In the ideal linear-Gaussian information-filter case, adding such a sensor cannot increase $\hat{\kappa}$. The real-world story is qualified: a biased, correlated, spoofed, delayed, or inconsistent sensor can raise the filter's posterior uncertainty (or, more often, raise the innovation-based consistency residual that the supervisor watches), and the framework's substrate runs a consistency check before admitting a new measurement into the joint update. Under that check, the monotonicity claim holds in the ideal limit and is guarded against the failure modes in production.

**The kid's analogy made precise.** The kid on the bike out of the tunnel has two information sources: pedals (high-rate, drifting) and phone (low-rate, bounded). The information-addition reading says: at every moment the kid's effective uncertainty about her position is set by both sources jointly. When the phone fix arrives, $\mathbf{I}_{\mathrm{fused}}$ jumps—the position uncertainty shrinks immediately. Between phone fixes, the INS bias contribution slowly grows the uncertainty, but never beyond the point where the next phone fix will pull it back. Her operational confidence—the framework's $C$—stays high as long as the phone fixes keep arriving on schedule. When she enters another tunnel, $\mathbf{I}_{\mathrm{gnss}}$ goes to zero, $\mathbf{I}_{\mathrm{fused}} = \mathbf{I}_{\mathrm{ins}}$ alone, and the operational confidence starts to drop with the integration time. This is the framework reading the multi-sensor fusion bundle through the kid's body.

§ 15.5

## *The framework in the substrate: GIGI and the multi-sensor bundle*

The ICARUS architecture's sensor-fusion implementation, already named in the verbatim sidebar of Chapter 10 §10.5, extends to multi-sensor fusion through the coproduct structure of §15.3.

**The substrate's bundle layout.** GIGI stores the coproduct bundle $E_{\text{fused}} = E_{\text{ins}} \sqcup E_{\text{gnss}} \sqcup \cdots$ as a single schema-declared object. The fiber tensor includes all sensor states, all bias states, and the joint covariance. The sensor thread runs the tightly-coupled Kalman update against this bundle at the rate of the fastest sensor (the IMU, typically 1 kHz). External sensor measurements (GNSS pseudoranges, magnetometer readings, altimeter readings) arrive at their own rates and are folded in at their respective measurement events.

**What the flight loop reads.** The flight loop, running at 200–400 Hz, reads sections of the fused bundle via GQL:

- `SECTION fused_state AT t = @latest_t` returns the most recent fused mean: position, velocity, attitude, and all sensor biases.
- `COVARIANCE fused_state AT t = @latest_t` returns the fused covariance $\mathbf{P}_{\text{fused}}$, used by downstream computations that need the uncertainty.
- `CURVATURE sensor_fusion` returns the substrate-computed $\hat{\kappa}$ for the fused bundle, used by the controller's gain modulation $m(C)$.

All three queries are constant-time. The flight loop never runs a Kalman update at flight time; the substrate runs it on the sensor thread, and the flight loop reads the results.

**The $\hat{\kappa}$ design choice for multi-sensor fusion.** The substrate's $\hat{\kappa}$ for $E_{\text{fused}}$ is the framework's standard design choice (Chapter 10 §10): a monotone non-negative function of $\text{tr}(\mathbf{P}_{\text{fused}})$, normalized against a baseline trace value. The baseline depends on which sensors are currently providing measurements; when a sensor drops out (GNSS in a tunnel, vision in the dark), the baseline shifts and the substrate signals reduced confidence to the controller. The controller's gain modulation $m(C)$ then throttles authority accordingly, exactly as in Chapter 8 §8.6.

**Why this is structurally interesting.** The fusion of any number of sensors is a single coproduct bundle with a single $\hat{\kappa}$. The controller does not need to know which sensors are currently active—it reads the substrate's confidence, and the substrate has already aggregated across the available sensors. New sensors can be added by extending the coproduct without modifying the controller's interface. This is the substrate-level modularity the framework is built around: the geometric structure (coproduct of bundles) is the engineering structure (modular sensor architecture), and changing one is changing the other.

§ 15.6

## *What you take from this chapter*

Four things.

**First.** GNSS and INS have complementary error structures by design: INS is smooth, high-rate, low-noise short-term, but

drifting long-term; GNSS is bounded long-term but intermittent, noisy, and unavailable under obstructions. Every modern navigation system pairs them through a Kalman filter that propagates the inertial state at high rate and corrects against external satellite measurements at the measurement rate.

**Second.** Three standard integration architectures exist: loosely coupled (GNSS produces a fix, INS uses it as a measurement), tightly coupled (GNSS pseudoranges go directly into the Kalman filter, with the receiver clock as a state), and deeply coupled (INS aiding closes the receiver's signal-tracking loops). ICARUS uses tight coupling.

**Third.** Multi-sensor fusion has a clean structural reading as a *coproduct in the category of fiber bundles.* Each sensor is a bundle of measurements over time; the coproduct is the union bundle whose fiber carries the information of all sensors; the Kalman filter operates on sections of the coproduct. Information-addition (in the Fisher sense) is the operational realization of the coproduct's universal property. Adding sensors extends the coproduct without modifying the architecture.

**Fourth.** The framework's design-choice $\hat{\kappa}$ for the sensor-fusion bundle, defined as a monotone non-negative function of $\mathrm{tr}(\mathbf{P}_{\mathrm{fused}})$, inherits the additive-information structure in the linear-Gaussian independent-measurement limit: more independent, correctly modeled, informative sensors means smaller trace, smaller $\hat{\kappa}$, higher operational confidence. Outside that limit (biased, correlated, delayed, or spoofed sensors), the substrate's consistency check guards the joint update against pathological increases. ICARUS reads this $\hat{\kappa}$ as a single substrate-computed scalar and gates the controller's gain modulation accordingly.

Chapter 16 closes Part V by extending the fusion framework one more step. In GNSS/INS integration, the external aiding signal (GNSS) is anchored in a known reference frame; the receiver

knows where the satellites are. In simultaneous localization and mapping (SLAM), the external aiding signal comes from features in an *unknown* environment, and the navigation filter has to estimate the map and the trajectory jointly. The bundle structure scales: SLAM is the coproduct of the navigation bundle and the map bundle, with the joint Kalman update operating on a state that includes both the vehicle's pose and the locations of every observed landmark. The kid riding through an unmapped neighborhood is doing SLAM whether she knows it or not.

*Vamos.*

Chapter 16

# *Simultaneous Localization and Mapping*

*The fundamental problem of SLAM is to use the relations between successive vehicle positions and feature locations to build a map and localize within it—simultaneously, from the same data.*
*Hugh Durrant-Whyte & Tim Bailey, "Simultaneous Localization and Mapping (SLAM): Part I," IEEE Robotics & Automation Magazine (2006)*

## § 16.0 *The kid in the unmapped neighborhood*

Imagine the kid is older now. Phone dead, no GPS. She is on her bike in a neighborhood she has never ridden through—borrowed bike, borrowed afternoon, no plan beyond *ride until something interesting happens*. The streets do not match the streets she knows. She is integrating felt motion forward from her starting point the way Chapter 14 described, and the integration is already

drifting the way Chapter 14 said it would.

But she has eyes.

A fire hydrant on the left corner of an intersection. A house with a porch swing and a magnolia tree. A corner store with a red awning. She does not have a map of this neighborhood, but she has a memory of what she has just seen, and she has the relative-distance estimate her pedaling gives her between one landmark and the next.

She rides for twenty minutes. The path twists; she takes turns based on what looks interesting. At some point she comes around a corner and there is the red awning again.

*Ya estuve aquí.* I have been here.

In that instant, three things happen at once. First, she knows the accumulated drift in her dead reckoning—the difference between where she thinks she is and where she actually is, which is the corner store she started near twenty minutes ago. Second, every landmark she has logged in the meantime gets corrected: each one's estimated position shifts by the same accumulated error that just got revealed. Third, she now has a closed loop on her mental map, and the map is sharper everywhere because of the loop, not just at the closure point.

This is **simultaneous localization and mapping**—SLAM, in the literature. The kid is doing it. She is estimating her own trajectory *and* the locations of every feature she has seen, jointly, from the same stream of proprioceptive integration and visual recognition. The remarkable thing is not that she is doing it. The remarkable thing is that a single local recognition—the red awning, again—propagates a global correction across her entire trajectory and her entire map.

The framework has a name for this propagation, and it has been hiding in plain sight since Chapter 1 §1.3: the **Bra Strap Principle**. A single local check, on a base space that is appropriately connected, determines the section globally (Davis 2026, Zenodo

DOI 10.5281/zenodo.20273573). Loop closure in SLAM is the principle's operational realization in the navigation-and-mapping domain. The kid recognizes the red awning; the propagation theorem does the rest.

This chapter walks SLAM the way Chapter 15 walked GNSS/INS: as the natural extension of the bundle framework one more summand outward. Chapter 14 gave us the inertial bundle. Chapter 15 added GNSS as a second summand in the coproduct. Chapter 16 adds the map itself as a third summand—the difference being that the map's contents are not known in advance, they are estimated from the same data that estimates the trajectory.

§ 16.1

## *State augmentation: the pose, the biases, and the map*

The navigation state vector from Chapter 15's tightly coupled integration contained: vehicle pose (6 dimensions), velocity (3), accelerometer and gyro biases (6), receiver clock and clock-drift offsets (2)—fifteen-and-change states, depending on the modeling. The Kalman filter propagated this state forward at IMU rate and corrected it at GNSS measurement rate.

SLAM augments the state vector with the map. Every landmark observed by the vehicle is appended as additional states:

$$\mathbf{x}_{\text{slam}} = \begin{pmatrix} \mathbf{x}_{\text{nav}} \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_N \end{pmatrix},$$

where $\mathbf{x}_{\text{nav}}$ is the standard navigation state and $\mathbf{m}_i \in \mathbb{R}^3$ is the estimated 3D position of the $i$-th observed landmark. The state vector grows as the vehicle observes new landmarks. A typical SLAM session might end with hundreds or thousands of landmarks in the state, depending on the duration and the density of recognizable features.

The bundle reading is the natural one. The navigation bundle from Chapter 14 has base = time and fiber = pose-velocity-biases. The map bundle has a different base: an index set $\{1, 2, \ldots, N(t)\}$ that grows with time, parameterizing the observed features. The map bundle's fiber over feature $i$ is the estimated position $\mathbf{m}_i$ together with its covariance and any descriptor (a visual signature, a radar return, an acoustic chirp).

The full SLAM bundle is the coproduct of the navigation bundle and the map bundle—the same structural move Chapter 15 §15.3 made for INS and GNSS, repeated with a different summand. The difference: GNSS's bundle was anchored in a known external frame (the satellites' published orbits); the map bundle is anchored in itself. The framework calls this difference the absence of external observability and reads it as a curvature property of the joint bundle's geometry, which is the subject of §16.2.

§ 16.2

## *The joint covariance and its cross-correlations*

The SLAM Kalman filter (the extended Kalman filter, or EKF, in its standard formulation) maintains a covariance matrix of the joint state. That matrix has block structure:

$$\mathbf{P}_{\text{slam}} \;=\; \begin{pmatrix} \mathbf{P}_{\text{nav,nav}} & \mathbf{P}_{\text{nav,map}} \\ \mathbf{P}_{\text{map,nav}} & \mathbf{P}_{\text{map,map}} \end{pmatrix}.$$

The diagonal blocks are familiar: $\mathbf{P}_{\text{nav,nav}}$ is the navigation covariance from Chapter 15, and $\mathbf{P}_{\text{map,map}}$ is the joint covariance of all landmark positions. The off-diagonal block $\mathbf{P}_{\text{nav,map}}$ is the chapter's main object of study.

**The cross-correlation block is the structure that makes SLAM work.** Every time the vehicle observes a landmark, the measurement equation links the landmark's position to the vehicle's pose:

$$\mathbf{z}_i \;=\; h(\mathbf{x}_{\text{nav}}, \mathbf{m}_i) \;+\; \mathbf{v}_i,$$

where $h(\cdot)$ is the observation model (a camera projection, a range-

bearing return, a stereo disparity) and $\mathbf{v}_i$ is the measurement noise. The Kalman update on this measurement does not merely correct the landmark or merely correct the pose. It corrects both jointly through the cross-correlation block, which the update step fills in with non-zero entries the first time the measurement is processed.

This is the reason landmark position estimates and pose estimates become correlated as a SLAM session progresses. The covariance is not a diagonal matrix that grows as landmarks are added. It is a fully populated joint covariance whose off-diagonal entries encode the inseparability of vehicle pose and landmark position once the two have been linked by an observation.

**The framework reading.** The cross-correlation block plays, structurally, the role that holonomy plays in the differential-geometric setting: both grow as the vehicle traverses paths in the base, both shrink when paths close, and both encode the integrated effect of the connection along the realized trajectory. The Ambrose-Singer machinery of Chapter 3 §3.3—which says that a connection's curvature is captured by the holonomy around small loops—is the geometric template the cross-correlation block instantiates operationally. A precise differential-geometric formulation of the cross-correlation block *as* a holonomy would require specifying the principal bundle, the connection, and the holonomy group with enough precision to make the identification rigorous; that formulation is open work. The structural reading is what the substrate operationalizes, and it is sufficient for the engineering use the chapter is making of it.

The condition number of $\mathbf{P}_{\text{slam}}$ is the operational quantity the framework cares about. When the condition number is low, the joint state is well-determined: the vehicle knows where it is, the map is sharp, the cross-correlations are tight. When the condition number rises, the joint state is becoming poorly determined: the vehicle is drifting along unobservable directions, the map is

drifting with it, the cross-correlations are loose. The framework's design-choice $\hat{\kappa}$ for SLAM is a monotone non-negative function of this condition number, in the same form Chapter 10 §10.4 used for sensor fusion.

§ 16.3

## *Loop closure as the Bra Strap Principle*

The single most operationally consequential event in a SLAM session is **loop closure**: the moment the vehicle re-observes a landmark it has seen before, after a long traversal in between. In the kid's neighborhood ride, this was the red awning showing up around the corner. In a robot's SLAM session, it is recognizing a previously logged feature from a sufficiently characteristic descriptor.

What happens at loop closure, mechanically, is one Kalman update. The re-observed landmark has an existing estimate $\hat{\mathbf{m}}_i$ in the map. The vehicle's current measurement $\mathbf{z}_i$ depends on the current pose and the landmark, through the same observation equation that produced the original estimate. The update applies this single new measurement to the joint state.

What happens at loop closure, *structurally*, is a global correction—with one important formulation-dependent caveat:

- In *EKF SLAM*, the joint state holds the current pose and all landmarks. The update step modifies the current pose and every landmark whose estimate is correlated with the pose (most of them, after a long traversal). The historical poses are not part of the state and so are not directly modified; their estimates are implicit in the trajectory's integrated motion model.

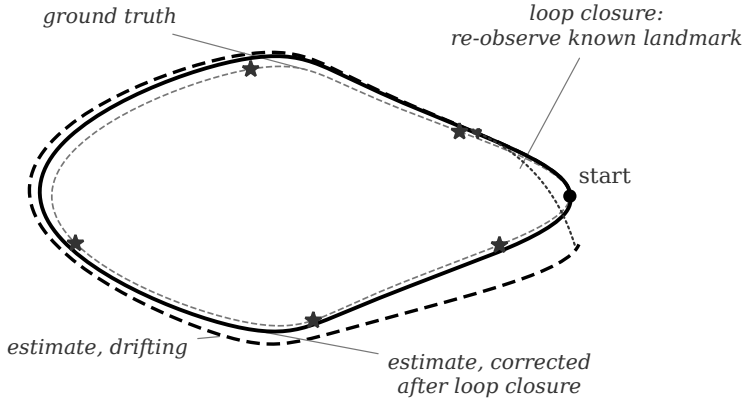



Figure 16.1: Loop closure in SLAM. The estimated vehicle trajectory (dashed) drifts away from the ground truth (light dashed) as the vehicle traverses unmapped terrain. At loop closure the vehicle re-observes a landmark logged earlier in the session (dotted arrow back to the known landmark). In a smoothing or factor-graph SLAM formulation, the least-squares update propagates this single local re-observation backward across the stored pose history and the entire map through the joint covariance's cross-correlation block; the corrected estimate of the trajectory (solid) snaps back toward the truth. In a pure EKF SLAM formulation, only the current pose and the map are updated, but the analogous structural pattern holds at each step.

- In *smoothing / factor-graph SLAM*, the state includes the full pose history. The single new measurement, applied as a factor between the current pose and the re-observed landmark, propagates a correction back across every pose and every landmark whose factor graph is connected to that node. Every landmark observed in the interval between the original sighting and the re-sighting has its estimate shifted in proportion to its correlation with the pose; the pose history is corrected jointly. Modern production SLAM systems use this formulation.

In both cases, a single new measurement triggers a state update whose support spans the structure the cross-correlation block links into a single inferential whole; the difference is whether that support includes the historical trajectory (smoothing) or only the current pose plus the map (EKF).

**This is the Bra Strap Principle made operational.** The principle, as Chapter 1 §1.3 stated it, says: on a base space that is appropriately connected, a single local check is sufficient to determine the section globally—provided the connection has the right propagation property. The base space, in SLAM, is the index set of poses and landmarks linked by observations. The local check is the re-observation of the awning. The connection's propagation property is the cross-correlation structure of the joint covariance. The conclusion is the global update.

The Davis Field Equation form $C = 1/(1 + \hat{\kappa})$ reads loop closure as the moment $\hat{\kappa}$ drops: a single new constraint, propagated through the connection, reduces the unobservable manifold's effective dimension and tightens the joint covariance. Maintaining the cross-correlation block is not unique to the Davis-Geometric framework—SLAM implementations have done this since Smith, Self, and Cheeseman in 1990. What the framework provides is a structural reading of *why* the cross-correlation block is the

right object to maintain: it is the operational form of the Bra Strap Principle's propagation property on the SLAM bundle. The substrate (ICARUS's GIGI) stores the block because the geometric reading and the empirical SLAM tradition agree on which structure has to be there for global propagation to work.

**The detection problem.** The Kalman update is the mechanical step. The harder problem is *detecting* that loop closure has occurred—deciding that the visual signature of the awning the vehicle just observed is in fact the same as the signature it logged twenty minutes ago, and not a different but similar awning. Modern SLAM systems use bag-of-words descriptors, learned feature embeddings, and explicit geometric consistency checks to decide. The detection step has its own failure modes: a false positive triggers a spurious global update that corrupts the entire map; a false negative misses a real loop closure and lets drift continue to accumulate. The framework's reading is that loop-closure detection is a separate confidence-bounded estimation problem nested inside the SLAM filter, with its own $\hat{\kappa}$ that gates whether the propagation step is allowed to fire.

§ 16.4

## *From EKF SLAM to pose-graph SLAM*

The extended Kalman filter is the historical formulation of SLAM, going back to Randall Smith, Matthew Self, and Peter Cheeseman's 1990 paper "Estimating Uncertain Spatial Relationships in Robotics," which posed the joint pose-and-map estimation problem in the form modern SLAM still uses. EKF SLAM is conceptually clean and has been the textbook entry point to the field ever since.

It does not scale.

The joint covariance has size $(6 + 3N) \times (6 + 3N)$ for $N$ landmarks, and the EKF update touches the full matrix at every step. The computational cost grows as $\mathcal{O}(N^2)$ per update and the memory grows as $\mathcal{O}(N^2)$ in total. For a SLAM session with ten thousand landmarks—which a modern visual SLAM system can produce in a few minutes of driving—the EKF approach is computationally untenable.

**The modern formulation is graph-based.** The state is no longer maintained as a joint covariance. It is maintained as a graph: nodes for each historical pose and each landmark; edges for each measurement that constrains the relationship between the entities it connects. The pose graph and the factor graph are two equivalent presentations of this object (Lu and Milios, "Globally Consistent Range Scan Alignment for Environment Mapping," *Autonomous Robots*, 1997; Grisetti, Kümmerle, Stachniss, and Burgard, "A Tutorial on Graph-Based SLAM," *IEEE Intelligent Transportation Systems Magazine*, 2010).

The estimation problem becomes a large sparse nonlinear least-squares problem: minimize the sum of squared measurement residuals over all the graph's edges, subject to the graph's connectivity. Modern solvers—`GTSAM`, `g2o`, `Ceres Solver`—exploit the sparsity through factorization-based methods that scale near-linearly in the number of measurements for typical SLAM graphs.

The terminology "bundle adjustment" is older than SLAM. It comes from photogrammetry in the 1950s, where the bundle being adjusted was the bundle of light rays from a 3D scene point through camera lenses to 2D image points. The mathematical problem—joint optimization of camera poses and 3D scene structure from observed image projections—is the same problem SLAM solves online. Modern visual SLAM systems are doing bundle adjustment with the additional constraint that the optimization has to run in real time and accommodate streaming

data.

**The framework reading.** The pose graph is the discretization of the SLAM bundle's section into a finite mesh. The factor graph is the dual presentation, where each measurement is a constraint on the section's value at the relevant nodes. The least-squares cost function is the integrated curvature of the section, in the framework's vocabulary; minimizing it is finding the section that is closest to flat (least surprise, most consistent with the measurements) given the graph's topology.

Loop closure in the graph formulation is the addition of a new edge between two existing nodes that previously had no direct edge. The new edge constrains the relative pose of those nodes through the loop-closure measurement; the optimizer propagates the constraint through every path connecting the two nodes; the global optimum shifts. This is the same Bra Strap propagation as in the EKF formulation, expressed in different mathematics.

§ 16.5

## *The framework reading: SLAM as coproduct with the map bundle*

Pulling the chapter's pieces together. The SLAM problem, in the framework's vocabulary:

**Base.** Time, augmented with a growing index set of observed landmarks. The base is no longer a fixed manifold; it is a manifold that admits new points as new landmarks are observed. The substrate's bundle storage architecture has to support this dynamic schema—GIGI's bundle layout does, by design.

**Fibers.** Over a time point, the fiber is the navigation state

(pose, velocity, biases). Over a landmark index, the fiber is the landmark's estimated position and descriptor. The total fiber over the full base is the direct sum of these two pieces.

**Sections.** A SLAM section is the estimated trajectory together with the estimated map. The section is what the filter (whether EKF or pose graph) is computing.

**Connection.** The connection 1-form is the joint output of the IMU (for time-axis transport) and the observation model (for landmark-axis transport). The strapdown algorithm transports the section along the time axis; the observation update transports the section along the landmark axis when a measurement links the two.

**Curvature.** The cross-correlation block of the joint covariance. Loop closures typically reduce its marginal uncertainty and redistribute correlations; long traversals without re-observation tend to grow it. (The block does not move monotonically: entries can grow, shrink, change sign, or redistribute depending on observability and graph structure.) The framework's $\hat{\kappa}$ for the SLAM bundle is a non-negative scalar derived from the whitened joint covariance—using at least two pieces of information, the scale (e.g. $\mathrm{tr}(\mathbf{\Sigma}_w)$ or $\log\det\mathbf{\Sigma}_w$) and the anisotropy ($\kappa(\mathbf{\Sigma}_w)$, the condition number of the whitened block)—and reports the resulting confidence scalar to the substrate. Condition number alone would not suffice: a covariance can be perfectly conditioned and still huge.

**Coproduct.** The SLAM bundle is the coproduct $E_{\mathrm{nav}} \sqcup E_{\mathrm{map}}$ in the category of bundles over the augmented base. Adding a new sensor (a LIDAR, an event camera) extends the coproduct one more summand. Adding a new landmark extends the map bundle's base by one point. The Kalman/least-squares update operates on sections of the coproduct, in the same way Chapter 15 §15.3 described for the simpler INS+GNSS case.

**The substrate.** GIGI stores the joint section, the joint covariance, and the cross-correlation block as a single queryable object. The flight controller reads $\hat{\kappa}_{\text{slam}}$ as a single substrate-computed scalar; the SLAM thread maintains the section against the measurement stream; the loop-closure detector runs as a sub-process whose own confidence gates whether the global update is allowed to fire. The architecture is the same one Chapter 15 §15.5 described for multi-sensor fusion, extended with the map bundle as an additional dynamic summand.

**ICARUS's role.** The controller does not see the SLAM internals. It sees $\hat{\kappa}_{\text{slam}}$ and applies the same curvature-gated gain modulation that Chapter 8 introduced and Chapter 20 formalizes. When the SLAM is observable and the map is sharp, $\hat{\kappa}_{\text{slam}}$ is small and the controller commands full nominal authority. When the SLAM is drifting along an unobservable manifold and the map is loose, $\hat{\kappa}_{\text{slam}}$ is large and the controller throttles down to the recovery-authority floor. The substrate handles the bookkeeping; the controller handles the response.

## § 16.6

## *What you take from this chapter*

Four things.

**First.** SLAM extends the navigation problem by adding the map itself to the estimated state. The Kalman filter (or its modern graph-based replacement) jointly estimates the vehicle's trajectory and the locations of all observed landmarks, from the same measurement stream that estimates either alone.

**Second.** The joint covariance has a cross-correlation block that links pose and map. Marginal uncertainty typically grows as

the vehicle moves and shrinks at loop closure, but the individual block entries are not monotone: they grow, shrink, redistribute, or change sign depending on which constraints fire and in which order. The block encodes the integrated curvature of the joint bundle along the trajectory; it is the structure that makes the propagation step possible.

**Third.** Loop closure is the Bra Strap Principle made operational. A single local re-observation propagates a global correction through the cross-correlation block: in smoothing / factor-graph SLAM, that propagation reaches the stored pose history as well as the map; in EKF SLAM, it reaches the current pose and every landmark correlated with it. Either way, the framework recognized the propagation requirement in advance and built the substrate to support it.

**Fourth.** The modern formulation is graph-based. EKF SLAM scales $\mathcal{O}(N^2)$ and is impractical at modern landmark counts; pose-graph SLAM scales near-linearly through sparse factorization and is the production formulation. Both are operating on the same underlying SLAM bundle; they differ in how the section is discretized and optimized.

This chapter closes Part V. Part V opened in a dark tunnel (§14.0) with the kid integrating felt motion forward without external reference. It moved to a sunlit open road (§15.0) with intermittent satellite anchoring. It closes in an unmapped neighborhood with the kid building the map herself, one recognized awning at a time.

The framework's structural reading of the navigation stack is now complete: navigation is parallel transport on a fiber bundle (Chapter 14); multi-sensor fusion is a coproduct in the category of bundles (Chapter 15); SLAM is the coproduct extended with a map summand whose base grows under observation (Chapter 16). Each step adds one structural piece. None of the pieces required

modifying what came before.

The kid who refused to ride the lawnmower-bike in Chapter 1 because she could not size her recovery authority is now riding through an unmapped neighborhood, building a map from a fire hydrant and a magnolia tree and a red awning, and the framework that lets her do it is the same framework that told her not to ride the bike. The bike was a closed-loop control problem with no recovery margin. The neighborhood ride is a closed-loop control problem *with* recovery margin—and now also with a map.

*Vamos.*

# *Lab IV: SLAM with loop closure — watching $\widehat{\kappa}$ drop at the awning.*

*Time: about 60–75 minutes. Materials: Python 3.10 or later with* `numpy`*,* `scipy`*,* `matplotlib`*, and* `dhoom`*.*

Lab III ran an adaptive law and watched a regressor's autocorrelation grow under PE. This lab runs an EKF-SLAM session on a synthetic 2D world, walks the joint pose-and-map covariance forward through a trajectory that leaves landmark range and returns, and lets you watch the substrate's $\widehat{\kappa}$ drop sharply at the loop-closure moment when the vehicle re-observes a landmark it logged earlier. The cross-correlation block from Chapter 16 §16.2 is the structure that does the work; this lab is where you see it work.

By the end you will have implemented EKF SLAM with proper inverse-Jacobian landmark initialization, driven a robot through an out-and-back trajectory with a U-turn between, observed the joint covariance's cross-correlation block grow during the traverse and shrink at loop closure, and stored the trajectory-and-map bundle as DHOOM.

**The world we'll use.** Five fixed landmarks near a straight outbound path. A robot with a unicycle motion model that drives 9 m straight east, performs a $\pi$-radian U-turn over 2 seconds,

then drives 9 m straight west along a path 1.27 m parallel to the outbound. An omnidirectional range-bearing sensor with 2 m range. Each landmark is observed in two separated intervals (outbound and inbound) with a substantial gap in between—a real loop closure.

Step 1

## *Set up*

```
mkdir lab4 && cd lab4
python  # or jupyter notebook

import numpy as np
import matplotlib.pyplot as plt
import dhoom

rng = np.random.default_rng(seed=42)
```

Step 2

## *The world: landmarks and the trajectory*

```
# Five landmarks near the outbound path
# (the kid's neighborhood features: fire hydrant, magnolia, red awning,
# mailbox, stop sign). The red awning is the canonical loop-closure
# target the chapter has been talking about.
landmarks_true = np.array([
    [2.0,  0.3],   # 0: fire hydrant
    [4.0, -0.4],   # 1: magnolia
    [5.0,  0.5],   # 2: RED AWNING (loop closure target)
    [7.0, -0.3],   # 3: mailbox
```

```
    [8.5,  0.4],   # 4: stop sign
])
n_landmarks = landmarks_true.shape[0]

# Trajectory: 9 m straight east, U-turn over 2 s, 9 m straight west.
n_steps = 200
dt = 0.1
v_cmd = 1.0
omega_cmd = np.zeros(n_steps)
omega_cmd[90:110] = np.pi / 2   # U-turn: 180 deg in 2 s

# Integrate the noise-free trajectory.
x_true = np.zeros((n_steps, 3))
for k in range(1, n_steps):
    x, y, th = x_true[k-1]
    x_true[k] = [x + v_cmd*np.cos(th)*dt,
                 y + v_cmd*np.sin(th)*dt,
                 th + omega_cmd[k-1]*dt]
```

Step 3

## *The motion model and measurement model*

```
# Motion model: unicycle with additive process noise.
sigma_v = 0.05      # m/s
sigma_omega = 0.03  # rad/s
Q = np.diag([sigma_v**2, sigma_omega**2])

def motion_model(x_prev, u, dt):
    x, y, th = x_prev; v, om = u
    return np.array([x + v*np.cos(th)*dt,
                     y + v*np.sin(th)*dt,
                     th + om*dt])

def motion_jacobian(x_prev, u, dt):
    _, _, th = x_prev; v, _ = u
    return np.array([[1.0, 0.0, -v*np.sin(th)*dt],
                     [0.0, 1.0,  v*np.cos(th)*dt],
                     [0.0, 0.0,  1.0]])
```

```
# Range-bearing sensor: omnidirectional (no FOV restriction),
# 2 m range. A 2 m range is short enough that a landmark goes
# OUT of view during the U-turn region and comes back during
# the inbound leg -- this is what makes loop closure observable.
sigma_r, sigma_b = 0.10, 0.05
R_meas = np.diag([sigma_r**2, sigma_b**2])
max_range = 2.0

def observe(x, lm):
    dx, dy = lm[0] - x[0], lm[1] - x[1]
    r = np.hypot(dx, dy)
    b = np.arctan2(dy, dx) - x[2]
    b = (b + np.pi) % (2*np.pi) - np.pi
    return np.array([r, b])

def visible(x, lm):
    r, _ = observe(x, lm)
    return r < max_range
```

Step 4

## *Generate noisy measurements along the true trajectory*

```
measurements = []
for k in range(n_steps):
    for j in range(n_landmarks):
        if visible(x_true[k], landmarks_true[j]):
            z = observe(x_true[k], landmarks_true[j])
            z += rng.multivariate_normal([0, 0], R_meas)
            measurements.append((k, j, z))
print(f'{len(measurements)} noisy measurements collected.')
```

Step 5

*Initialize the SLAM state and covariance*

```
# Joint state: [x_robot (3), lm0_x, lm0_y, lm1_x, lm1_y, ..., lm4_x, lm4_y]
# Dimension: 3 + 2 * n_landmarks = 13.
N = 3 + 2 * n_landmarks
mu = np.zeros(N); mu[:3] = x_true[0]
P = np.eye(N) * 1e6              # landmarks: huge initial uncertainty
P[:3, :3] = np.diag([0.01, 0.01, 0.005])   # robot pose: small uncertainty

observed_landmarks = set()
mu_hist = np.zeros((n_steps, N))
kappa_hist = np.zeros(n_steps)
cross_corr_norm = np.zeros(n_steps)
```

Step 6

*EKF predict and update — with proper landmark covariance initialization*

```
def predict(mu, P, u, dt):
    F = np.eye(len(mu))
    F[:3, :3] = motion_jacobian(mu[:3], u, dt)
    mu_new = mu.copy()
    mu_new[:3] = motion_model(mu[:3], u, dt)
    G = np.zeros((len(mu), 2))
    G[0, 0] = np.cos(mu[2]) * dt
    G[1, 0] = np.sin(mu[2]) * dt
    G[2, 1] = dt
    P_new = F @ P @ F.T + G @ Q @ G.T
    return mu_new, P_new

def update(mu, P, z, lm_idx, observed_landmarks):
    lm_offset = 3 + 2 * lm_idx
    if lm_idx not in observed_landmarks:
```

```
        # First sighting: initialize BOTH mean AND covariance via
        # inverse-observation Jacobian (Bar-Shalom et al., Estimation
        # with Applications to Tracking and Navigation, section 6.5).
        # Without this, the new landmark block stays at the 1e6 prior
        # and kappa_hat would JUMP UP at first sighting.
        r, b = z
        x, y, th = mu[:3]
        ang = b + th
        c, s = np.cos(ang), np.sin(ang)
        mu[lm_offset]   = x + r * c
        mu[lm_offset+1] = y + r * s
        J_z = np.array([[c, -r*s], [s,  r*c]])
        J_p = np.array([[1.0, 0.0, -r*s], [0.0, 1.0,  r*c]])
        P_pose = P[:3, :3]
        P_mm = J_p @ P_pose @ J_p.T + J_z @ R_meas @ J_z.T
        idx = [lm_offset, lm_offset + 1]
        P[np.ix_(idx, idx)] = P_mm
        P[:3, idx] = P_pose @ J_p.T
        P[idx, :3] = P[:3, idx].T
        if observed_landmarks:
            other_idx = [3 + 2*j + k
                         for j in observed_landmarks
                         for k in (0, 1)]
            P[np.ix_(idx, other_idx)] = J_p @ P[:3, other_idx]
            P[np.ix_(other_idx, idx)] = P[np.ix_(idx, other_idx)].T
        observed_landmarks.add(lm_idx)
        return mu, P
    # Standard EKF update for a previously-observed landmark.
    z_pred = observe(mu[:3], mu[lm_offset:lm_offset+2])
    dx = mu[lm_offset]   - mu[0]
    dy = mu[lm_offset+1] - mu[1]
    q = dx*dx + dy*dy
    sq = np.sqrt(q)
    H = np.zeros((2, len(mu)))
    H[0, 0] = -dx/sq;  H[0, 1] = -dy/sq;  H[0, 2] = 0.0
    H[0, lm_offset]   =  dx/sq;  H[0, lm_offset+1] =  dy/sq
    H[1, 0] =  dy/q;   H[1, 1] = -dx/q;   H[1, 2] = -1.0
    H[1, lm_offset]   = -dy/q;   H[1, lm_offset+1] =  dx/q
    y_inn = z - z_pred
    y_inn[1] = (y_inn[1] + np.pi) % (2*np.pi) - np.pi   # wrap bearing
    S = H @ P @ H.T + R_meas
    K = P @ H.T @ np.linalg.inv(S)
    mu = mu + K @ y_inn
    P = (np.eye(len(mu)) - K @ H) @ P
    return mu, P
```

Step 7

## *Run SLAM with normalized $\hat{\kappa}$*

```
# Index measurements by step.
meas_by_step = {k: [] for k in range(n_steps)}
for (k, j, z) in measurements:
    meas_by_step[k].append((j, z))

# Run the SLAM and record histories.
for k in range(n_steps):
    if k > 0:
        u_k = np.array([v_cmd, omega_cmd[k-1]])
        mu, P = predict(mu, P, u_k, dt)
    for (j, z) in meas_by_step[k]:
        mu, P = update(mu, P, z, j, observed_landmarks)
    mu_hist[k] = mu
    if len(observed_landmarks) > 0:
        cross_corr_norm[k] = np.linalg.norm(P[:3, 3:], 'fro')

# Normalize kappa_hat against the trace at the LAST timestep that
# corresponds to the substrate's steady-state observation. Using
# the trajectory's median tr(P) as the baseline keeps kappa_hat
# dimensionless and centered around zero in nominal operation.
# This is the same kappa_hat design choice Lab V uses (Chapter 10
# section 10.4 family).
def kappa_hat_normalized(P, observed, baseline):
    obs_idx = [0, 1, 2]
    for j in observed:
        obs_idx += [3 + 2*j, 4 + 2*j]
    tr = float(np.trace(P[np.ix_(obs_idx, obs_idx)]))
    if baseline <= 0:
        return tr
    return max(0.0, tr / baseline - 1.0)

# Compute the baseline as the median trace of the observed block
# during the steady-state portion of the inbound leg (steps 130-200,
# after the U-turn and after the system has had time to settle into
# its post-loop-closure regime).
baseline_indices = list(range(130, n_steps))
traces = []
mu_tmp = np.zeros(N); mu_tmp[:3] = x_true[0]
P_tmp = np.eye(N) * 1e6
P_tmp[:3, :3] = np.diag([0.01, 0.01, 0.005])
```

```
observed_tmp = set()
for k in range(n_steps):
    if k > 0:
        u_k = np.array([v_cmd, omega_cmd[k-1]])
        mu_tmp, P_tmp = predict(mu_tmp, P_tmp, u_k, dt)
    for (j, z) in meas_by_step[k]:
        mu_tmp, P_tmp = update(mu_tmp, P_tmp, z, j, observed_tmp)
    if k in baseline_indices and observed_tmp:
        obs_idx = [0, 1, 2]
        for j in observed_tmp:
            obs_idx += [3 + 2*j, 4 + 2*j]
        traces.append(float(np.trace(P_tmp[np.ix_(obs_idx, obs_idx)])))
tr_baseline = float(np.median(traces)) if traces else 1.0
print(f'kappa_hat baseline'
      f' (median tr P over steady-state): {tr_baseline:.4f}')

# Recompute kappa_hist using the normalized form.
mu_run = np.zeros(N); mu_run[:3] = x_true[0]
P_run = np.eye(N) * 1e6
P_run[:3, :3] = np.diag([0.01, 0.01, 0.005])
observed_run = set()
mu_hist = np.zeros((n_steps, N))
for k in range(n_steps):
    if k > 0:
        u_k = np.array([v_cmd, omega_cmd[k-1]])
        mu_run, P_run = predict(mu_run, P_run, u_k, dt)
    for (j, z) in meas_by_step[k]:
        mu_run, P_run = update(mu_run, P_run, z, j, observed_run)
    mu_hist[k] = mu_run
    kappa_hist[k] = kappa_hat_normalized(P_run, observed_run, tr_baseline)
    if len(observed_run) > 0:
        cross_corr_norm[k] = np.linalg.norm(P_run[:3, 3:], 'fro')

print(f'kappa range: [{kappa_hist.min():.3f}, {kappa_hist.max():.3f}]')
```

Step 8

## *Plot the four panels*

```
fig, axes = plt.subplots(2, 2, figsize=(11, 8))
```

```
ax = axes[0, 0]
ax.plot(x_true[:, 0], x_true[:, 1], 'k-', alpha=0.4, label='true')
ax.plot(mu_hist[:, 0], mu_hist[:, 1], 'b-', label='estimated')
ax.scatter(landmarks_true[:, 0], landmarks_true[:, 1],
           c='red', marker='*', s=150, label='true landmarks')
for j in observed_run:
    ax.scatter(mu_run[3+2*j], mu_run[4+2*j], c='blue', marker='+', s=100)
ax.set_xlabel('x [m]'); ax.set_ylabel('y [m]')
ax.legend(); ax.set_aspect('equal'); ax.set_title('Trajectory and map')

ax = axes[0, 1]
ax.plot(np.arange(n_steps)*dt, kappa_hist, 'b-')
ax.set_xlabel('time [s]'); ax.set_ylabel(r'$\widehat{\kappa}$')
ax.axhline(0, color='gray', alpha=0.5, linestyle='--')
ax.set_title(r'Substrate curvature scalar')
ax.grid(True)

ax = axes[1, 0]
ax.plot(np.arange(n_steps)*dt, cross_corr_norm, 'g-')
ax.set_xlabel('time [s]'); ax.set_ylabel(r'$\|P_{\mathrm{nav,map}}\|_F$')
ax.set_title('Cross-correlation block (Chapter 16 section 16.2)')
ax.grid(True)

ax = axes[1, 1]
for j in range(n_landmarks):
    if j in observed_run:
        err = np.sqrt(
            (mu_hist[:, 3+2*j] - landmarks_true[j, 0])**2 +
            (mu_hist[:, 4+2*j] - landmarks_true[j, 1])**2)
        first_seen = next(k for (k, j2, _) in measurements if j2 == j)
        err[:first_seen] = np.nan
        ax.plot(np.arange(n_steps)*dt, err, label=f'lm {j}')
ax.set_xlabel('time [s]'); ax.set_ylabel('position error [m]')
ax.legend(); ax.set_title('Landmark position error')
ax.grid(True)

plt.tight_layout(); plt.savefig('lab4_results.png', dpi=120)
plt.show()
```

## *What you should see*

Three things should be visible in the plots.

**One: $\widehat{\kappa}$ rises during outbound exploration, peaks during the U-turn, drops sharply during inbound loop closures.** During the outbound leg (steps 0–90), each landmark gets initialized in turn, each first sighting adding a finite-variance block to the joint covariance with the inverse-Jacobian initialization the lab implements. Between sightings the predict step grows the joint covariance through process noise. $\widehat{\kappa}$ rises monotonically during this phase. The U-turn region (steps 90–110) sees the robot lose sight of all landmarks momentarily; $\widehat{\kappa}$ peaks here as the predict-step uncertainty accumulates without measurement correction. The inbound leg (steps 110–200) re-observes every landmark on the second pass, applying loop-closure-style updates that tighten the joint covariance through the cross-correlation block; $\widehat{\kappa}$ drops sharply at each re-observation event. This is the Bra Strap Principle of Chapter 1 §1.3 running operationally—each re-observed landmark propagates a global correction across the entire joint state through the cross-correlations.

**Two: the cross-correlation block grows during outbound and shrinks during inbound.** The Frobenius norm of the pose-map cross-correlation block traces a clean inverted-V shape: growing during outbound, peaking during the U-turn, shrinking during inbound. This is the holonomy reading from Chapter 16 §16.2 instantiated on a real run: the cross-correlation grows as the vehicle traverses paths in the bundle's base and shrinks when those paths close.

**Three: landmark position errors stabilize after first observation, then improve further on re-observation.** Each

landmark's position estimate has a typical error of a few centimeters after first sighting (consistent with the measurement noise levels). On re-observation during the inbound leg, the errors tighten further as the cross-correlation block propagates the corrected pose estimate back to each landmark.

Step 10

## *Store the run as DHOOM*

```
# DHOOM bundles require exactly one top-level key (the bundle name).
# The records are a flat list, one per timestep.
slam_records = []
for k in range(n_steps):
    rec = {
        'k': k,
        't': float(k * dt),
        'rx': float(mu_hist[k, 0]),
        'ry': float(mu_hist[k, 1]),
        'rth': float(mu_hist[k, 2]),
        'kappa': float(kappa_hist[k]),
    }
    for j in range(n_landmarks):
        rec[f'lm{j}_x'] = float(mu_hist[k, 3 + 2*j])
        rec[f'lm{j}_y'] = float(mu_hist[k, 4 + 2*j])
    slam_records.append(rec)
slam_bundle = {'slam_run': slam_records}
with open('lab4_slam_run.dhoom', 'w') as f:
    f.write(dhoom.encode(slam_bundle))
print('SLAM run serialized to lab4_slam_run.dhoom')
```

The serialized bundle preserves the joint trajectory-and-map section together with the substrate's curvature reading at every timestep, as a flat list of per-timestep records. Loading this bundle into a future GIGI instance lets you query the run, run `COMPLETE` against the sparse stretches, and `CONSISTENCY`-check

the section against the bundle's coboundary residual—the diagnostic Chapter 18 §18.3 names (which, in well-conditioned cellular-sheaf cases, reads as the dimension of the first sheaf cohomology $H^1$).

This closes Part V. The next part walks the architecture that runs SLAM (and the rest of the navigation stack) at flight time.

Chapter 17

# *The Substrate Principle*

*The connections between modules are the assumptions which the modules make about each other.*
*David L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," Communications of the ACM (1972)*

§ 17.0

## *The bike you didn't grow up on*

A friend lets the kid borrow her bike. The friend is a year older. The bike is a year bigger—longer top tube, taller saddle, a different brand of brake lever, the chain ring a couple of teeth larger than what the kid is used to. The kid gets on and tests the brakes, lifts her weight onto the pedals, listens for the chain skip, and rides.

She does not need to know the brand. She does not need to read the spec sheet. Her body's control loop is reading a small handful of things—the feel of the brakes under her fingers, the lever throw before the pads bite, the responsiveness of the rear

wheel to her hip shift, the threshold at which the front wheel feels light. These are the variables the controller reads. The bike's identity, its country of manufacture, its model year, the alloy of its frame, the bearing race on the crank—none of these reach the controller. They are below the interface.

She rides the friend's bike for an hour. When she trades back, she gets on her own bike and her body adapts again, in ten seconds, to the geometry it has been adapting to since she was twelve. The control loop is invariant. The bike is the substrate. The interface between them is narrow: the brakes work or they don't, the front end feels light or it doesn't, the chain is engaging or skipping. A small number of well-typed signals carry everything the controller needs.

This chapter is about that interface, taken seriously, as an architectural principle. It is the principle that the book has been operating under for sixteen chapters without naming explicitly. Naming it now lets the rest of Part VI walk the specific software realization that this book is about: the flight-control architecture I have been calling **ICARUS**, the *Integrated Curvature-Aware Routing and Uncertainty Substrate*, U.S. Provisional Patent Application 64/045,891, filed April 21, 2026, with the corresponding fiber-bundle database engine **GIGI** as U.S. Provisional 64/045,889 (same filing date) and the serialization layer **DHOOM** as U.S. Provisional 64/008,940 (filed March 18, 2026).

The patents are the formal record. The book is the geometric reading of why the architecture has the shape it has. Part VI walks the architecture as the operational realization of the framework from Parts I–V.

§ 17.1

## *Part VI: what this part covers*

Part VI is the architecture section. Parts I–V gave the geometric reading: bundles, sections, connections, holonomy, the Davis Field Equation, multi-sensor coproducts, SLAM with map bundles. Part VI gives the software realization: how the geometric machinery is held in memory, queried at flight time, and used to gate the controller's commands.

**Chapter 17 (this chapter)** states the architectural principle—the *Substrate Principle*—and introduces ICARUS as its canonical instance.
**Chapter 18** walks GIGI, the fiber-bundle database engine that holds the substrate's state. The chapter shows how bundles, sections, connections, and cross-correlation blocks are stored, indexed, and queried.
**Chapter 19** walks DHOOM, the serialization format that carries bundle data across processes, networks, and persistent storage without losing the geometric type information.
**Chapter 20** integrates the three: the full ICARUS architecture, the GQL query language, the lock-free fast path, the supervisor's stability proofs, and the worked example of a quadrotor running the entire stack end-to-end.

Part VI does not introduce new mathematics. It explains how the mathematics from Parts I–V is realized in code that runs on actual flight hardware. The reader who has carried the geometric vocabulary through sixteen chapters now sees that vocabulary become a set of concrete data structures, function signatures, and concurrency primitives—and sees that the concrete realization preserves the geometric structure it implements.

§ 17.2

## *The Substrate Principle*

The Substrate Principle, stated formally:

*The flight controller does not maintain state. The substrate maintains state. The controller reads from the substrate and writes commands to it. The interface between controller and substrate is a small set of well-typed queries, the most important of which returns the framework's confidence scalar $\hat{\kappa}$.*

This is the Parnas information-hiding principle, applied to flight control with the substrate-as-interface drawn at the specific boundary the geometric reading recommends. The substrate hides three classes of information from the controller:

**Sensor identity.** The controller does not know which sensors are currently active, which have failed, which were never installed, or which are being processed at what rate. The substrate handles all sensor management internally.

**Bundle structure.** The controller does not know whether the substrate is running INS-only navigation, INS+GNSS coproduct, full SLAM with map bundle, or any other multi-bundle configuration. The substrate handles the bundle architecture internally.

**Estimator internals.** The controller does not know whether the substrate is using an extended Kalman filter, an unscented filter, a particle filter, or a pose-graph optimizer. The substrate handles estimation internally.

What the controller sees, instead, is a small set of queries that the substrate guarantees to answer at the controller's read rate:

- `state()` returns the current best estimate of the pose, velocity, and any biases the controller needs.

- `covariance()` returns the joint covariance of those estimates (or a chosen marginal).
- `curvature()` returns the framework's design-choice $\hat{\kappa}$.
- `command(u)` accepts the controller's commanded actuator vector and forwards it to the actuators with whatever curvature-gated modulation the substrate has computed for the current cycle.

That is the interface. Four queries. The substrate guarantees that each is constant-time and lock-free at the controller's read rate, which on ICARUS's reference implementation is 1 kHz.

The interface's narrowness is the architectural point. Anything wider couples the controller to substrate internals; anything narrower fails to carry the information the framework's chosen modulation policy needs. The geometric reading is what tells us that $\hat{\kappa}$ is the right scalar to carry on the interface *for the curvature-gated modulation policy of Chapter 8 §8.6*: under that policy the gain modulation is a function of $\hat{\kappa}$ alone, so $\hat{\kappa}$ is the sufficient interface signal. Other safety-critical decisions—directional thrust allocation, anisotropic actuator authority, sensor-fault routing—require richer signals than a single scalar, and the substrate publishes those through separate, equally narrow interfaces tailored to each consumer's policy.

§ 17.3

## *Why one scalar suffices for the gain-modulation policy*

The reader unfamiliar with the architectural literature might reasonably ask: why is $\hat{\kappa}$ enough for the controller's gain-modulation

decision? Why does the gain modulator not need the full covariance, the bundle structure, the sensor health flags, the loop-closure quality, the model uncertainty estimate, and a dozen other signals?

The answer is scoped: $\hat{\kappa}$ is sufficient *for the conservative gain-modulation policy* the framework has chosen, not for every safe control decision a flight system has to make. Chapter 8 §8.6 defines the modulation as $m(\hat{\kappa}) \in [m_{\min}, 1]$, a function of $\hat{\kappa}$ alone, and proves that this modulation delivers a strictly positive recovery-authority floor for any $\hat{\kappa}$ in its domain. The proof requires only that $\hat{\kappa}$ be a monotone non-negative function of the joint covariance's conditioning. It does not require any specific covariance entries, any specific bundle structure, or any specific estimator type. *For this policy*, one scalar is enough. Decisions outside the policy—thrust allocation under saturation, anisotropic actuator authority, estimator-inconsistency response, sensor-fault classification—are made against richer substrate-published signals, each carried on its own narrow, well-typed interface.

**This is the framework earning its keep.** The geometric reading—curvature obstruction as a non-negative scalar, gain modulation as a monotone function of curvature—is what makes the gain-modulation interface narrow. Without the geometric reading, the controller's modulation logic would have to inspect the substrate's internals to know what gain to use, because there would be no principled scalar to read. With the geometric reading, the substrate computes one number and the modulator reads it. Other consumers (the thrust allocator, the sentinel's proximity logic, the supervisor's Lyapunov monitor) read different substrate-published signals for their own decisions; $\hat{\kappa}$ is the narrow interface for the gain-modulation consumer specifically.

The principle generalizes. Any future sensor, any future bundle architecture, any future fusion algorithm: as long as it produces

an $\hat{\kappa}$ that satisfies the monotone-non-negative property, the controller does not change. The architecture absorbs the change at the substrate boundary.

§ 17.4

## *Reader threads, writer threads, and the fast path*

The substrate is concurrent. Multiple writer threads update the substrate's state from sensor measurements:

**IMU writer**: high-rate (1 kHz), low-latency, runs the strapdown integration from Chapter 14.
**GNSS writer**: low-rate (1–10 Hz), variable latency, runs the tight-coupling Kalman update from Chapter 15.
**Vision writer**: medium-rate (30–60 Hz), variable latency, runs the SLAM observation update from Chapter 16.
**Telemetry writer**: variable, runs the logging and the ground-station mirror.

Multiple reader threads consume the substrate's state at their own rates:

**Controller reader**: high-rate (1 kHz), runs the LQR gain modulation and the actuator command.
**Planner reader**: low-rate (1–10 Hz), runs trajectory generation and waypoint following.
**Supervisor reader**: medium-rate, runs the envelope-protection logic and the Lyapunov-monitor described in Chapter 11.

**The architectural challenge.** The controller cannot block on a sensor update. If the GNSS writer is mid-update when the controller reads, the controller must see a consistent snapshot of the substrate's previous state—not a partially-updated state that

mixes new pose with old covariance. The controller's correctness requires causally complete snapshots at every read.

ICARUS implements this through *lock-free atomic snapshots*. The substrate maintains a small ring buffer of completed snapshots. Each writer commits a new snapshot atomically when its update is complete; the snapshot becomes the new current state. Readers take the snapshot pointer at the start of their read and use that snapshot for the duration of the read, even if a writer commits a newer snapshot in the meantime. The reader sees a consistent state; the writer is never blocked by a reader.

This is the *fast path* the codebase names. The Rust adapter's `fast_path` module implements the lock-free snapshot mechanism for the highest-rate consumers. The reference implementation achieves $< 10\,\mu$s controller-read latency under nominal load and $< 50\,\mu$s under worst-case contention.

§ 17.5

## *Graceful degradation as substrate invariance*

The Substrate Principle's most consequential operational consequence is *graceful degradation under sensor failure*.

A scenario. The vehicle is in cruise. GNSS works, IMU works, vision works. $\hat{\kappa}$ is small; the controller is at nominal gain; the vehicle responds crisply to commands. A GNSS receiver loses lock—a building obstruction, a constellation gap, a multipath event. The substrate notices: the GNSS writer stops committing new measurements, the SLAM writer's loop-closure rate drops because the absolute-position prior is gone, the joint covariance grows along directions that GNSS had been constraining.

**What the substrate does.** The substrate's $\hat{\kappa}$ rises. The

covariance's conditioning is worse, so the framework's design-choice non-negative scalar—computed from the conditioning, as Chapter 10 §10.4 prescribed—takes a larger value. The substrate continues to publish a consistent state estimate; it is just a state estimate with larger covariance.

**What the controller does.** The controller reads the larger $\hat{\kappa}$ at its next cycle. The curvature-gated modulation $m(\hat{\kappa})$ takes a smaller value, monotone in $\hat{\kappa}$, bounded below by $m_{\min}$. The controller applies the reduced gain. The vehicle's commanded acceleration is smaller, the closed-loop bandwidth is lower, the recovery margin is preserved.

**What the controller does not do.** It does not know the GNSS receiver failed. It does not know that vision is now the dominant position aiding. It does not know that the SLAM filter has dropped to a lower update rate. It reads one scalar—$\hat{\kappa}$—and modulates accordingly.

This is what the architecture delivers. Failure modes manifest as $\hat{\kappa}$ rising. Recovery manifests as $\hat{\kappa}$ falling. The controller's behavior is invariant under the substrate's internal reconfiguration; the substrate's internal reconfiguration is invariant under the controller's gain modulation. Each side handles its own concerns through the narrow interface.

The vehicle degrades gracefully because the architecture was built to make graceful degradation the default response, not a special case. The kid in the driveway sized her recovery authority before riding the lawnmower-bike, Chapter 1 §1.0; the architecture sizes its recovery authority continuously, every control cycle, through the interface the Substrate Principle defines.

§ 17.6

## *What you take from this chapter*

Four things.

**First.** The flight controller does not maintain state. The substrate maintains state. The controller reads from the substrate and writes commands. This is the *Substrate Principle*, the architectural commitment the book has been operating under since Chapter 1 and is naming formally now.

**Second.** The interface between controller and substrate is narrow by design: a small set of well-typed queries, the most important of which returns the framework's design-choice confidence scalar $\hat{\kappa}$. For the curvature-gated modulation policy of Chapter 8 §8.6, $\hat{\kappa}$ is the sufficient interface scalar; other consumers read other narrow interfaces tailored to their policies.

**Third.** The substrate is concurrent. Writer threads (IMU, GNSS, vision, telemetry) update the substrate's state; reader threads (controller, planner, supervisor) consume it. ICARUS uses lock-free atomic snapshots so that high-rate readers never block on writes and never see partially-updated state. The Rust adapter's `fast_path` module is the reference implementation.

**Fourth.** Sensor failure manifests at the controller as $\hat{\kappa}$ rising. The controller modulates gain accordingly through the curvature-gated modulation $m(\hat{\kappa})$, monotone non-increasing in $\hat{\kappa}$, bounded below by $m_{\min} > 0$. The vehicle degrades gracefully. The controller does not know which sensor failed; it knows that $\hat{\kappa}$ moved.

Chapter 18 walks GIGI, the fiber-bundle database engine that holds the substrate's state and answers the controller's queries. The chapter shows how the substrate's data structures preserve

the geometric typing of bundles, sections, connections, and cross-correlation blocks—and shows the GQL query language that the controller and the planner use to read the substrate without breaking the interface this chapter has been about.

*Vamos.*

Chapter 18

# *GIGI: The Fiber-Bundle Database*

*Cell and tissue, shell and bone, leaf and flower, are so many portions of matter, and it is in obedience to the laws of physics that their particles have been moved, moulded and conformed.*
*D'Arcy Wentworth Thompson, On Growth and Form (1917)*

§ 18.0

## *Teddy's ears*

I was at the kitchen table working on a serialization format. The dog was on the floor next to my chair. His name is Teddy. He is a long-haired mutt with the kind of ears that, when he turns his head, sweep a small arc through the air and resettle in a slightly different configuration than the one they started in. He had been there for a while. I had been staring at a JSON document for a while. At some point I looked down.

Teddy's ears.

The strands. Hundreds of them coming off each ear, each one its own length, its own curl, its own angle of departure from the skin. Different from every other strand. But all of them attached at the base. The base was the ear—a curved surface, a connected manifold of skin and cartilage—and the strands were what stuck out of it. Every point on the base had its strand. Every strand was different. The whole arrangement was held together not by repetition but by *attachment*: each strand belonged to the place on the base where it was rooted, and the place determined which strand it was.

*Eso es.*

That is a fiber bundle. Not a metaphor. Literally. A fiber bundle in the differential-geometric sense—base manifold, projection, fiber at each base point. The dog's ear is the canonical example. I had been reading about fiber bundles for years; I had used them in my flight-control work; I had written a paper about them; I was, at that moment, holding the kind of recognition the kid in the driveway had held when she saw that a chain ring and a lawnmower blade were both circles in motion. I was looking at a dog and seeing the structure my serialization format was supposed to encode.

The serialization format was DHOOM. The recognition that came next was bigger than DHOOM. If nature organizes information as fiber bundles—if every dog and every leaf and every coastline is doing this—then the question is not just *how do you serialize a fiber bundle.* The question is *how do you* ***store*** *one.* Not in transit, not over the wire, but at rest, in a database, queryable, indexed, fast.

This chapter is the answer. GIGI is a database engine whose primary data structure is the fiber bundle. U.S. Provisional Patent Application 64/045,889, filed April 21, 2026, alongside ICARUS. The chapter walks what GIGI is, why a fiber-bundle-

native database is qualitatively different from the relational, document, columnar, and search-index databases that came before, and how GIGI's central capability—*sheaf completion*—is the operational realization of the Davis Duality (Davis 2026, Zenodo DOI `10.5281/zenodo.19428406`) at the storage layer.

§ 18.1

## *DHOOM: what happens when a differential geometer looks at JSON*

The chronology matters. DHOOM came first. GIGI was the question DHOOM made unavoidable.

DHOOM started on LinkedIn. People were arguing about `TOON`—a recently proposed JSON-replacement that factors out repeated field names into a header. The arguments were the usual arguments: JSON is verbose, TOON is opinionated, schemas should be external, schemas should be inline, LLMs read tokens not characters, and so on. I read the arguments and thought: *I can beat TOON.*

The reason I thought I could beat TOON is that I had been working with fiber bundles for a long time, and JSON's verbosity has a specific shape when you read it as a bundle. JSON repeats the field names on every record. TOON factors them into a header. Both formats are still transmitting redundancy that the fiber-bundle reading lets you quotient out. The redundancy is geometric: the field names are a property of the *base* (the schema), not of the *fibers* (the individual records). Once you recognize that, you can encode the base once and let the fibers be the only data you transmit.

DHOOM—*Davis Human-readable Optimized Object Markup*, U.S.

Provisional 64/008,940, filed March 18, 2026—is what comes out when you take that recognition seriously across every kind of structural redundancy JSON contains. Each of DHOOM's syntactic markers corresponds to a geometric operation:

- `|default` declares a constant fiber: a value the receiver assumes unless overridden. Silence means agreement.
- `>` declares a child base inherits its parent's schema. Nested names are not repeated.
- `^` declares delta-encoded fibers: each fiber is transmitted as a difference from a reference fiber, the way connection coefficients are encoded as departures from a flat metric.
- `~` declares a sparse bundle: nulls are not transmitted, only the populated fibers.
- `->` declares a morphism between bundles: a cross-reference whose semantics is a typed map.
- `&` declares an associated bundle of interned strings: a separate fiber bundle whose fibers are reused by integer index.
- `#` declares a computed field: a section that is determined by other sections through a defined formula. The receiver computes; the sender does not transmit.
- `!` declares an inline constraint: the section conditions a section has to satisfy.

The principle is one sentence: *don't transmit what the receiver can derive.* The principle is geometric: anything the bundle's structure determines is already at the receiver as soon as the structure is. On the same review-data example DHOOM's documentation uses, JSON is 412 characters, TOON is approximately 210, DHOOM is approximately 137. The character savings are not the point. The point is that the savings come from honoring

the data's actual geometric structure, not from cleverer text packing.

§ 18.2

## *From DHOOM to GIGI: could the storage layer be fiber-bundle-native?*

Once DHOOM existed, the next question was unavoidable. If serialization can be fiber-bundle-native, why not storage? Every database in production—SQL, Cassandra, Elasticsearch, Druid, MongoDB, Snowflake, the entire landscape—stores data in one of a small number of representations: rows, documents, columns, time-series. Each representation is a flat-tangent special case of a fiber bundle. Rows fix the schema (the fiber type) and let you query a single row at a time. Columns fix the column (a single coordinate of the fiber) and let you query across rows. Documents soften the schema constraint but still embed the fiber's structure in a tree. None of them store the bundle itself: the base manifold, the connection, the fibers attached to the base, the geometric relationship between fibers at nearby base points.

GIGI does.

**What GIGI is, structurally.** A GIGI database is a collection of named fiber bundles. Each bundle has:

- A *base*—an indexed set of base points, with a topology defined by an explicit *adjacency* relation. The base can be 1D (time), 2D (a spatial grid), graph-structured (social networks), or higher-dimensional manifolds. Adjacency is what defines which base points are *near* each other in the connection's sense.

- A *fiber schema*—a typed record of what is attached at each base point. The schema is the type the fiber must satisfy; individual fibers can be sparse against the schema and GIGI tracks which fields are populated.
- A *connection*—a transport rule defining how to move a fiber from one base point to an adjacent one. The connection can be the trivial connection (just copy the fiber over), an affine transport, a Rodrigues rotation (the `TRANSPORT_ROTATION` GQL verb provides $N$-dimensional rotation transport in closed form), or a custom transport defined by the user.
- A *section*—the actual stored data: a fiber-value at each populated base point.

Queries against a GIGI bundle are queries against sections of that bundle. The GIGI Query Language (GQL) lets you ask, for example, `SELECT velocity AT t IN [t0, t1]` (a section over a time interval), `TRANSPORT pose FROM t0 TO t1` (parallel-transport the pose along the time-axis connection), or `COVARIANCE state OVER bundle_join(ins, gnss)` (the joint covariance of a coproduct, in the sense of Chapter 15 §15.3).

**What this buys you that the existing databases do not.** Bundles compose. A multi-sensor fusion architecture—which Chapter 15 described as a coproduct in the category of fiber bundles—is a first-class GIGI operation, not an application-layer afterthought. A SLAM session—which Chapter 16 described as a coproduct with a dynamically-growing map summand—is a GIGI bundle whose base extends as the vehicle observes new landmarks. The bundle's connection (the IMU's measurement equation, the camera's observation model) is stored alongside the data, not in application code that wraps the data.

The competitive frame: SQL handles flat tables. Cassandra handles wide rows. Elastic handles search indexes. Druid handles

time-series. Each is a special case of *a single base point's flat tangent space*. GIGI handles the base manifold and the connection between base points. Graph databases (Neo4j, Neptune, Dgraph) and array databases (TileDB, SciDB) carry richer base structure than the four named—adjacency relations and dense base manifolds, respectively—and they are closer cousins to GIGI than the relational landscape. What still distinguishes GIGI is the *connection as a first-class object*: an explicit transport rule between fibers at adjacent base points, with the sheaf-Laplacian solver of §18.3 built into the engine. Without the connection, sheaf completion is not available; with the connection, it is the engine's primary operation. The earlier databases are recoverable as flat-base or connection-less GIGI configurations; the connection-bearing GIGI is not recoverable from any of them, because none of them carry that piece of structure.

§ 18.3

## *Sheaf completion:* `COMPLETE`*,* `PROPAGATE`*,* `CONSISTENCY`

This is the section the chapter has been building toward. It is the capability that, when I built it and ran it for the first time on a sparse bundle with missing fibers, made me realize GIGI was not a faster database. It was a different kind of object.

Standard databases return what you stored. If you query a row that does not exist, you get `NULL`. If you query a column with missing values, you get `NULL` for those values. The database's job ends at the boundary of what was written.

GIGI's job does not end there.

**The `COMPLETE` verb.** Given a sparse section—fibers populated

at some base points, missing at others—GIGI returns the completion: the section that satisfies the bundle's structural axioms at every base point, computed by solving the *sheaf Laplacian* on the adjacency graph. The Laplacian's solve is implemented as a Schur complement reduction: the known fibers become a boundary condition, the unknown fibers become the variables, and the solution is the section closest to flat (least integrated curvature) given the constraints.

This is the discrete-sheaf-cohomology machinery applied in a production database. The Laplacian is the discrete sheaf Laplacian on the bundle's adjacency graph. The Schur complement is the standard linear-algebraic shortcut for solving a sparse linear system with a known partition. The completion is not interpolation in the statistical sense—it does not assume smoothness, it does not fit a regression, it does not invoke a Gaussian process prior. It solves the harmonic-extension equation the bundle's connection defines: the unique least-curvature section consistent with the stored boundary fibers.

**The `PROPAGATE` verb.** Given a known fiber at one base point, transport it along the connection to a chosen target base point. This is the Bra Strap Principle made operational at the storage layer: a single local value, propagated through the connection's path, determines the value at any target the connection's adjacency graph can reach (Davis 2026, Zenodo DOI `10.5281/zenodo.20273573`). Where `COMPLETE` solves the global system from boundary data, `PROPAGATE` pushes a single fiber along a chosen path.

**The `CONSISTENCY` verb.** Given a section that is overdetermined—fibers at many base points, each constraining the others through the connection—GIGI returns a *consistency residual*: the $\ell^2$ norm of the coboundary (equivalently, the least-squares energy of the stored data against the bundle's

connection, computed via the sheaf Laplacian). A residual of zero means the stored data glues consistently: there is no obstruction to the completion, and `COMPLETE` will succeed exactly. A non-zero residual means the data has obstructions: somewhere in the bundle, the stored fibers contradict each other under the connection, and `COMPLETE` returns the least-squares projection onto the space of consistent sections rather than an exact solution. In the special case of a well-conditioned cellular sheaf with the standard cochain complex, this residual is directly related to (the dimension of) the first sheaf cohomology $H^1$; for general bundle structures the residual is the operational diagnostic, and the cohomological reading is its structural interpretation. GIGI samples the residual on large bundles where exact computation is infeasible; the sampling estimator plays the same structural-diagnostic role at the storage interface that the framework's $\hat{\kappa}$ plays at the controller interface.

**Why this matters operationally.** The three verbs are not utilities bolted onto a relational core. They are the primary operations the engine is built to perform. A query against a GIGI bundle is, by default, a query against the completion—you ask for a fiber at a base point, and if the fiber is stored, you get it; if it is not stored but the bundle's structure determines it, GIGI computes it and returns it with the structural confidence $\hat{\kappa}$ that the curvature of the path to the nearest stored data implies.

This is what made me realize, the first time I ran it on actual flight data with a deliberately dropped sensor stream, that the architecture was not faster than the alternatives. It was answering a different question.

§ 18.4

## *The geodesic and the flat map*

The framework reading of what is happening in §18.3 is the one I have been pointing at since Chapter 1.

Statistical machine learning, in its supervised-regression core, is the operation *estimate the conditional expectation of a target variable given features.* For linear regression and the classical kernel methods, the conditional expectation is literally a flat-tangent approximation: a linear (or fixed-basis) function of the features that minimizes mean-squared error against the training data. The tangent-space reading is precise in this regime. For modern deep learning, the structural reading is looser: a neural network is a composition of nonlinear function classes, and the universal-approximation theorems show that sufficiently wide or deep networks can approximate arbitrary continuous functions on compact domains, which is stronger than tangent-space projection at every point. What deep learning shares with the linear case is the optimization target—minimize an empirical loss over a parametric class against training data—and the structural failure mode that follows from that target, which is what the next paragraph names.

A geodesic is what the bundle's connection actually traces. When you parallel-transport a vector along a path on a curved manifold, the vector does not arrive at the destination by linear extrapolation. It arrives along the trajectory the connection prescribes, and the difference between where the linear extrapolation predicts it and where it actually arrives is the integrated curvature along the path.

**The Davis Duality reading.** The Davis Duality of Approximation and Obstruction (Davis 2026, Zenodo DOI `10.5281/zenodo.19428406`) gives the two-sided bound: the error of a flat

approximation against a curved section is bounded above *and below* by a multiple of the integrated curvature times the step size squared. Above means the flat approximation is not arbitrarily bad. Below means it cannot be made arbitrarily good either. The lower bound is the news: there is a structural error floor that no amount of clever fitting can get below.

Statistical ML lives within this floor. It is right where the curvature is low—where flat-tangent or locally-flat approximations are a good fit to the underlying data manifold. It is fragile, in a specific structural way, in regions of high curvature or under distribution shift: classical linear models fail there outright, and nonlinear ML methods can represent curved structure only when the training data covers those regions densely enough that the fitted function class can be regularized to match them. Modern ML practitioners experience this as *distribution shift*, *adversarial brittleness*, *calibration failure in the tail*: the model that works on the bulk of the data fails when the input lands somewhere the local linearization no longer matches the manifold.

GIGI does not approximate *when the connection is known.* `COMPLETE` solves the discrete harmonic-extension problem that the bundle's specified connection induces (the sheaf-Laplacian system), returning the least-curvature section consistent with the stored boundary data. This is not the geodesic equation in the differential-geometric sense—a second-order ODE for a path on a manifold—but its discrete elliptic-PDE analogue on the bundle's adjacency graph, which is the operational object the storage layer can actually solve. When the connection is the correct one and the stored fibers are dense enough that the Laplacian solve is well-conditioned, the completion is the actual section. When the connection is correct but the data is sparse, GIGI returns the completion *with* $\hat{\kappa}$—the same scalar the substrate publishes to the controller—telling the consumer how much to trust the result. When the connection itself is uncertain (the user has

declared a connection that may not match the true structure of the data), the framework's commitment is unchanged but its operational meaning shifts: GIGI is now answering "what would the completion be *under this connection*?" and the consumer carries the responsibility of choosing the connection that matches the data's actual structure.

**The connection back to Sameness.** This is the picture from *The Geometry of Sameness* (Davis, December 1, 2025) the book opened with. Statistical ML is the flat map. The geodesic is the globe. Mercator's projection is right at the equator and wrong at the poles, in exactly the way the Davis Duality predicts. ML is right where the data manifold is locally flat and wrong where the data manifold curves. GIGI's contribution is not to fix ML. It is to store data in a form where the curvature is preserved, queryable, and computable, so that the consumer of the data can know which regime they are in before they reach for a flat method.

The Davis Duality is the unity. Approximation and obstruction are the two sides of the same equation: $C = \tau/K = 1/(1+\hat{\kappa})$, the Davis Field Equation in its operational form. `COMPLETE` reads the equation as a solver; statistical ML reads it as a regressor; the framework reads them as the same equation, evaluated in different regimes of the curvature.

§ 18.5

## *GIGI in the wild: PRISM, MIRADOR, MARCELLA, ICARUS, DEMETER*

GIGI is not a research prototype. It runs in production behind a series of named applications, each of which uses the bundle substrate for its own domain.

**ICARUS** is the flight control architecture this book is about. The substrate stores the navigation bundles, the SLAM map bundles, the controller-state bundle, and the cross-correlation blocks the Kalman update maintains. The controller's $\hat{\kappa}$ is a GIGI-computed scalar; the actuator commands are GIGI-stored sections.

**MIRADOR** is the architectural review system. It reads the bundle structure of GIGI itself—commits, dependency graphs, test coverage, API surface area—and audits proposed changes against the bundle's geometric invariants. When a commit would introduce an inconsistency in the section structure (a breaking API change, a coverage regression, a constraint violation), MIRADOR raises $\hat{\kappa}$ on the relevant bundle and surfaces the issue. The recurring "MIRADOR review fixes" commits in the GIGI repository are the trace of this self-auditing loop.

**PRISM** is the data-product front-end: a tenant-aware view layer over GIGI bundles, used to render structured data to downstream consumers (dashboards, embedded analytics, federated queries) without the consumer needing to know the underlying bundle structure.

**MARCELLA** is a transformer-replacement architecture for language models (Chapter 21). Token embeddings are a section of a fiber bundle over a token-position base; attention is parallel

transport on that bundle; the residual stream is the horizontal lift along the predicted trajectory of meaning. The state-dependent composite connection $\Gamma = \alpha\,\Gamma_{\text{topic}} + \beta\,\Gamma_{\text{voice}} + \gamma\,\Gamma_{\text{identity}} + \delta\,\Gamma_{\text{state}}$ transports the prompt's intent along a chosen path; the path's holonomy is the compositional residue—meaning that wasn't in any of the source sections, accumulated by the connection along the loop the path traces. The implementation is in the `fiber_lm` package; Chapter 21 walks the architecture in detail.

**DEMETER** is the agricultural and environmental data system: spatial bundles over geographic base manifolds, with sheaf completion used to fill in observations where ground stations are sparse and the connection (driven by terrain, weather, and crop models) determines the unobserved sections.

Five applications, one substrate. The substrate's commitment from Chapter 17—narrow interface, controller-invariant under sensor changes, $\hat{\kappa}$ as the single confidence scalar—generalizes from flight control to every other domain where a bundle structure is the right reading of the data. The applications do not share code beyond GIGI; they share the geometric reading of what their data is.

§ 18.6

## *What you take from this chapter*

Four things.

**First.** DHOOM (U.S. Provisional 64/008,940) is fiber-bundle-aware serialization. Each of its syntactic markers corresponds to a geometric operation; the format quotients out the redundancy that the bundle's structure already determines. The principle is

*don't transmit what the receiver can derive*; the geometric reading is what tells you which derivations the receiver can perform.

**Second.** GIGI (U.S. Provisional 64/045,889) is the fiber-bundle database engine. Bundles are first-class data structures with base, schema, connection, and section. Queries traverse base manifolds. Parallel transport is a verb. The substrate that Chapter 17 described is the GIGI engine running underneath the flight controller's interface.

**Third.** The central capability is *sheaf completion*: `COMPLETE` solves the discrete sheaf-Laplacian system via Schur complement to recover missing fibers as the least-curvature harmonic extension; `PROPAGATE` transports a single fiber along the connection's adjacency graph; `CONSISTENCY` returns a coboundary-based residual that diagnoses inconsistent stored data (and that, in well-conditioned cellular-sheaf cases, computes the dimension of the first sheaf cohomology $H^1$). These are GQL verbs, not utilities. They are the primary operations the engine is built to perform.

**Fourth.** The Davis Duality reading is the unity. Statistical machine learning is the flat-tangent approximation to the data manifold; GIGI's `COMPLETE` is the geodesic computation on the same manifold. ML is right where the curvature is low and wrong where it is high; GIGI is right where the bundle's structure is densely stored and reports its own $\hat{\kappa}$ where it is not. The Field Equation $C = \tau/K = 1/(1 + \hat{\kappa})$ governs both. ML reads the equation in the flat regime. GIGI reads it across the regimes.

Chapter 19 walks DHOOM in detail: the wire format, the parser, the geometric profile API, and the benchmarks against JSON and TOON. Chapter 20 integrates the substrate's storage layer (GIGI) and the serialization layer (DHOOM) with the controller and the supervisor into the full ICARUS stack, with the quadrotor worked example.

*Vamos.*

Chapter 19

# *DHOOM in detail*

*Don't transmit what the receiver can derive.*
*The DHOOM specification, v0.5 (2026)*

§ 19.0

## *The directions you don't have to send*

I am texting a friend. We have known each other for years; we have lived in Sacramento longer; we share enough context that the shorthand works. We are meeting for dinner at a place neither of us has been to. I do not text: *take I-5 South, exit at Florin Road, turn left at the light, continue two blocks, the restaurant is on the right.* I text: *Florin Road, left at the light, two blocks.* She fills in I-5, she fills in South, she fills in which light, she fills in which side of the street. She fills in the context because she has the context. Anything she has, I do not transmit.

This is the principle DHOOM is built on, stated in one sentence: *don't transmit what the receiver can derive.* The principle is older than software. Telegrams used codebooks because every word was billed. Radio operators developed Q-codes because every second of airtime mattered. Sign language compresses

ideas into gestures because the body has only so many degrees of freedom. In every case, the sender and receiver share enough structure that the message can quotient out the redundancy and still arrive intact.

Chapter 18 introduced DHOOM as the serialization layer of the bundle substrate. This chapter walks the format in detail—wire syntax, the eight quotient operations each marker performs, the geometric profile API, and the benchmark numbers against JSON, TOON, and the tokenizers modern language models read with.

§ 19.1

## *The wire format*

A DHOOM document begins with a *header*: a single line declaring the schema of the records that follow. The header is the base manifold of the bundle being serialized—it tells the receiver what type of fiber is attached at each base point.

A minimal header for a list of reviews:

```
reviews[N]{id, customer, rating, comment, verified}:
```

The bracketed `[N]` declares the cardinality (how many records follow, or `[*]` for unknown-length streams). The brace-delimited list declares the fields. After the colon, the records appear one per line, in the field order the header established. Field names are transmitted once.

A record line is a comma-separated tuple of values, in the header's order:

```
101, Alex Rivera, 5, Excellent!, true
```

On this baseline—header-once, records as positional tuples—DHOOM matches TOON's compression. The eight syntactic markers walked in §19.2 are what take DHOOM further, by recognizing that the record-tuple representation still carries redundancy that the bundle's structure can quotient out.

§ 19.2

## *The eight quotient operations*

Each DHOOM marker performs a specific quotient against a specific kind of redundancy. The geometric reading of each is given alongside.

`|default`—declares a constant fiber. The field's value, unless overridden, is the declared default. Silence in a record means agreement with the default. Geometrically: this is a flat section—a constant function on the base—and the receiver reconstructs the constant from the header alone.

`@start`—declares a sequential index derived from position. The first record gets `@start`, the second gets `@start+1`, and so on. The field is not transmitted in any record; the receiver derives it from position. Geometrically: this is the identity section on a one-dimensional base, encoded by its starting offset.

`>`—declares a child base inherits its parent's schema. A nested object whose fields match the parent's omits the field names. Geometrically: this is base inheritance—the child bundle's base is the parent bundle's base, restricted to the relevant subset.

`^`—declares a delta-encoded field. Each record's value is the difference from the previous record's value. Useful for time-series, monotone IDs, and slowly-varying signals. Geometrically: this

is the connection one-form's representation in coordinates—the connection coefficients $\Gamma$ that describe how the section moves from one base point to the next, rather than the section's values themselves.

`~`—declares a sparse bundle: nulls are not transmitted, only the populated fibers. Each record's sparse fields are listed by name, the rest are silence. Geometrically: this is a sparse section—the base points where the section is supported, with the unsupported region encoded by absence.

`->`—declares a morphism between bundles: a typed cross-reference whose semantics is a structure-preserving map. A field that points to another bundle's record is encoded as a morphism, not as a duplicated copy. Geometrically: this is a bundle morphism in the categorical sense—a map of bundles over the same base that commutes with projection.

`&`—declares an associated bundle of interned strings. A string pool is built once; each record references the pool by integer index. Useful when the same string appears in many records. Geometrically: this is an associated bundle—a separate fiber bundle whose fibers (the strings) are referenced from the primary bundle by integer-valued sections.

`#`—declares a computed field whose value is determined by other fields through a formula declared in the header. The receiver computes the value; the sender does not transmit it. Geometrically: this is a section over a quotient bundle—the field is the projection of the joint state onto a derived coordinate, recoverable from the joint state alone.

`!`—declares an inline constraint: a condition the section must satisfy at each base point. The receiver validates; the sender does not need to repeat that the constraint holds. Geometrically: this is the constraint manifold the section lies on—the bundle's local condition that defines which fibers are admissible.

Each marker is one syntactic character. Each character corresponds to one geometric operation. The format's compactness is a side effect of the format's geometric honesty.

§ 19.3

## *The Profile API: geometric data analysis on the wire*

DHOOM ships with a *Profile API*: a small set of operations that read a DHOOM document and return geometric properties of the bundle it encodes. The Profile API does not just describe the data; it describes the data's structure as a bundle.

**Cardinality and sparsity.** The number of populated base points, the number of populated fibers per base point, and the sparsity ratio (populated fields divided by total fields). These are the bundle's first-order metrics.

**Field entropy.** For each field, the entropy of its values across records. Low entropy means the field is a near-constant (a candidate for `|default`). High entropy means the field carries genuine information.

**Cross-field correlation.** The pairwise correlation between fields. Highly correlated fields are *candidates* for `#` (computed-field) elision, but correlation alone is not sufficient: a correlation coefficient of 0.99 still leaves 1% of the field's variance unexplained, and eliding the field without a deterministic formula loses that residual. The Profile API surfaces correlation as a hint; the schema author still has to supply an explicit `#` formula plus a validation residual (the maximum tolerated difference between the formula's output and the original value) before the field is

actually elided. The API recommends; it does not unilaterally drop information.

**Delta efficiency.** For each field, the relative variance of consecutive differences versus the variance of values. High delta efficiency means the field is a candidate for `^` (delta encoding).

**Adjacency structure.** If the base supports an adjacency relation, the Profile API estimates the connection's regularity—how much each fiber differs from its neighbors. This is the operational analogue of the coboundary-based residual that GIGI's `CONSISTENCY` verb (Chapter 18 §18.3) computes at the storage layer—the diagnostic that, in well-conditioned cellular-sheaf cases, computes the dimension of the first sheaf cohomology $H^1$, and that in general bundle structures plays the operational role.

The Profile API is what makes DHOOM a tool for *discovering* the right compression markers rather than just *applying* them. A user with a JSON document and no schema can run the Profile API and get back a proposed DHOOM header with markers chosen to maximize compression against the document's specific redundancy pattern.

§ 19.4

## *Benchmarks: characters, tokens, retrieval*

DHOOM publishes three sets of benchmarks. Each measures a different property of the format, and each matters for a different consumer.

**Character count.** The traditional measure: how many bytes of UTF-8 the document occupies. On the published review-data

example (three records, five fields), JSON minified is 412 characters, TOON is approximately 210, DHOOM is approximately 137. The savings widen as the data grows: sparse wide tables with repeated string values compress by 80% or more against JSON.

**Token count (GPT-4o tokenizer, o200k_base).** The modern measure: how many tokens a language-model tokenizer produces from the document. Tokens, not characters, are what language models charge for and what they fit in their context windows. DHOOM's syntactic markers were chosen to tokenize compactly: `|default`, `@start`, `^`, `~`, `->`, `&`, `#`, `!` each consume one or two tokens in the o200k_base tokenizer. The token-count compression ratio is consistently 30–60% better than JSON.

**LLM retrieval accuracy.** The benchmark that closes the loop. DHOOM's specification publishes a 209-question retrieval benchmark against Claude Sonnet, comparing the model's question-answering accuracy on JSON, TOON, and DHOOM representations of the same data. The result: DHOOM does not lose retrieval accuracy at its lower token count. The compression preserves the information the model can extract.

This last benchmark is what made DHOOM more than a format. It demonstrated that the geometric reading—the quotient operations the markers perform—preserves the data's queryable content under the tested retrieval workload. Compression that loses retrieval accuracy is just lossy compression. Compression that preserves retrieval accuracy is structure-preserving, and that is what the bundle reading licenses in advance: anything that is preserved by the *declared deterministic* quotient operations is, by definition, recoverable from the quotient. The empirical retrieval result extends this to model-mediated retrieval on the benchmark corpus and model used; the structural guarantee covers only the deterministic recoverability piece, not arbitrary LLM extraction

behaviour.

§ 19.5

## *When to use DHOOM (and when not to)*

DHOOM is not the right format for everything. The format's compression depends on the data having geometric structure—a shared schema, redundant fields, sparse populations, delta-able sequences, or interned strings. Data without these properties does not benefit.

**Where DHOOM helps.** Tabular records with a shared schema. Time-series with monotone or slowly-varying fields. Sparse wide tables. Log records with repeated keys. Sensor streams with stable field structure and varying values. API responses with consistent shape. The flight-control telemetry that ICARUS publishes is precisely this kind of data; DHOOM compresses ICARUS telemetry by 60–80% against JSON, depending on the bundle.

**Where DHOOM does not help.** Unstructured text. Binary blobs (images, audio, sensor recordings). One-off documents without repeated structure. Data with high entropy in every field. Documents whose readers are humans, not machines, and where readability of the original JSON outweighs the compression. DHOOM is a format for structured data being read by software; for free-form text it offers nothing.

**Where DHOOM is currently best deployed.** As the wire format between ICARUS components, between GIGI nodes in a federation, and between the substrate and downstream consumers (dashboards, analytics, replay tooling). The book's claim is not

that DHOOM should replace JSON everywhere. The claim is that, for the kind of bundle-structured data the framework produces, DHOOM is the natural serialization layer because it preserves on the wire what GIGI preserves at rest and what the controller relies on at runtime.

§ 19.6

## *What you take from this chapter*

Four things.

**First.** The DHOOM principle is one sentence: *don't transmit what the receiver can derive.* The principle is geometric: anything the bundle's structure determines is already at the receiver and does not need to be sent. The format's markers are the syntactic realization of this principle.

**Second.** Each of the eight markers (`|default`, `@start`, `>`, `^`, `~`, `->`, `&`, `#`, `!`) performs a specific geometric quotient against a specific kind of redundancy. The markers are not encoding tricks; they are typed operations on the bundle's structure.

**Third.** The DHOOM Profile API reads a document and returns the bundle's geometric properties—cardinality, sparsity, field entropy, cross-field correlation, delta efficiency, adjacency regularity. The API enables *discovering* the right compression markers from a sample of the data rather than requiring the schema author to choose them by hand.

**Fourth.** Three benchmark axes: character count, token count (o200k_base tokenizer), and LLM retrieval accuracy. DHOOM compresses by 30–80% on character and token counts against

JSON, and the retrieval-accuracy benchmark shows that the compression preserves the queryable content of the data. Structure-preserving compression is what the geometric reading guarantees.

Chapter 20 integrates DHOOM, GIGI, and the controller into the full ICARUS stack, with the quadrotor worked example end-to-end. The serialization, the storage, the substrate interface, and the controller's response all sit in the architecture that the kid in the driveway, Teddy on the kitchen floor, and the bike out of the tunnel have been preparing the reader to see.

*Vamos.*

Chapter 20

# *ICARUS, integrated*

*Architecture is the thoughtful making of space.*
*Louis I. Kahn, lecture at the Yale School of Architecture (1962)*

§ 20.0

## *The kid is the architecture*

Twenty chapters in, the architecture the book has been pointing at is on the table. ICARUS is the integrated flight-control system whose name has appeared in every chapter and whose components have been walked one at a time: the framework's geometric reading (Parts I–II), the controller's gain modulation (Part III), the estimator and the adaptive layer (Part IV), the navigation stack (Part V), the Substrate Principle (Chapter 17), the bundle database (Chapter 18), the serialization format (Chapter 19).

This chapter closes the loop by putting all of it in one place.

The kid in the driveway in Chapter 1 §1.0 chose not to ride because she could not size her recovery authority. The kid through twenty chapters has been doing the same operation in

different domains: feel the brakes before riding the new bike, integrate felt motion forward through the tunnel, recognize the awning at loop closure, borrow a friend's bike and adapt within ten seconds. Every one of those operations is the architecture this chapter formalizes. The kid's sensorimotor controller is reading a confidence scalar from a fast-updating substrate of bundle-structured state and modulating her actions through a recovery-authority floor. She has been doing this since she was a child. ICARUS is the version that runs in software.

§ 20.1

## *The ICARUS stack, end to end*

ICARUS—U.S. Provisional Patent Application 64/045,891, filed April 21, 2026—is a flight-control architecture in five layers.

**Layer 1: Sensors and actuators.** The physical interface. IMU, GNSS receiver, vision system, barometric altimeter, magnetometer on the input side; rotor speed commands or control-surface deflections on the output side. The actuator commands are what eventually leave the system as control authority; everything else exists to compute them.

**Layer 2: GIGI substrate.** The bundle database from Chapter 18. Holds the navigation bundle, the SLAM map bundle, the controller-state bundle, the sensor-fusion coproduct, and the cross-correlation blocks that link them. Maintains the joint section against incoming sensor measurements; computes the confidence scalar $\hat{\kappa}$; serves the controller's queries at the controller's read rate.

**Layer 3: DHOOM wire format.** The serialization layer from Chapter 19. Carries bundle data between the substrate and

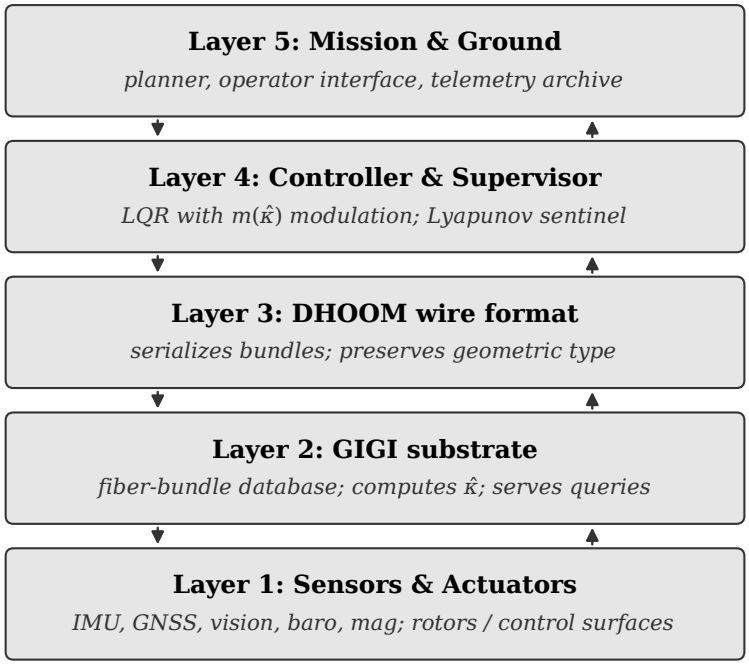



Figure 20.1: The ICARUS architecture, five layers. Sensors and actuators at the physical bottom; the GIGI substrate above them holding the joint section and computing $\hat{\kappa}$; the DHOOM wire format above that, serializing bundles between processes; the controller and supervisor above that, reading state and writing commands; and the mission / ground layer at the top. All cross-layer communication flows through GIGI's narrow substrate interface; no layer reaches across another layer's boundary.

external consumers (telemetry, replay tooling, federated GIGI nodes) and between internal processes that do not share memory. Preserves the bundle's geometric type information on the wire.

**Layer 4: Controller and supervisor.** The control logic. The controller (LQR with curvature-gated modulation $m(\hat{\kappa})$ from Chapter 8) reads $\hat{\kappa}$ and the relevant state, computes the actuator command, and writes it back to the substrate. The supervisor (Lyapunov-monitor and envelope-protection logic from Chapter 11 and the sentinel module) watches the closed loop, checks the envelope conditions, and emits advisories if the system is approaching a boundary.

**Layer 5: Mission and ground.** The planner, the operator interface, the ground station. Sets waypoints, accepts operator overrides, displays telemetry, archives flight data. This layer is outside the closed-loop control system; its interface to ICARUS is through GIGI's external API.

The layers communicate through the Substrate Principle's narrow interface (Chapter 17): each layer reads from the substrate and writes to it, with no direct cross-layer coupling. Layer 4 does not know which sensors Layer 1 is currently using. Layer 2 does not know which controller Layer 4 is running. Layer 3 does not know what the data means; it only knows the bundle's type.

§ 20.2

## *The controller's GQL queries*

At each control cycle (1 kHz on the reference implementation), the controller issues a small set of GQL queries against GIGI. The queries are constant-time and lock-free; they return the

state the controller needs and accept the command the controller produces.

The actual query set, simplified for exposition:

```
SECTION pose AT t = @latest_t
SECTION velocity AT t = @latest_t
SECTION biases AT t = @latest_t
CURVATURE sensor_fusion
COMMAND actuator <- u
```

The first three queries read the current state estimate—pose (position and attitude), velocity, and current bias estimates—from the substrate's latest committed snapshot. The fourth queries the curvature scalar $\hat{\kappa}$ for the active sensor-fusion bundle (Chapter 15 §15.5). The fifth submits the controller's commanded actuator vector to the substrate, which forwards it to the actuators with whatever curvature-gated modulation has been computed for the current cycle.

Five GQL operations per control cycle. Each is on the order of 1–10 microseconds on the reference Rust implementation (the `icarus-adapter` crate's HTTP/WebSocket client to GIGI, with the `fast_path` module providing the lock-free shortcut for the highest-rate operations). The total controller-tick latency stays comfortably under the 1 ms budget that the 1 kHz control rate imposes.

The queries are typed against the bundle schema. The compiler that translates them into runtime calls verifies, at build time, that the controller is asking for fields that the substrate is configured to provide, in the right units, with the right covariance structure. A controller written against the wrong schema is a compile-time error, not a runtime failure. The framework's geometric typing extends from the substrate's data structures all the way to the controller's source code.

§ 20.3

## *The supervisor and the sentinel*

The controller is the inner loop. The supervisor and the sentinel are the outer loops that watch the inner loop.

**The supervisor.** Implements the Lyapunov-monitor argument from Chapter 11 in its discrete-time, noise-tolerant form: maintains a candidate Lyapunov function on the closed-loop state, evaluates it at each cycle, and checks the ultimate-boundedness condition $\Delta V(\mathbf{x}_k) \leq -\alpha\|\mathbf{x}_k\|^2 + \eta$, where $\alpha > 0$ sets the contraction rate and $\eta$ is a tolerance for sample noise, disturbance, and modeling error. Strict monotonic decrease at every tick is the wrong test for a sampled, noisy, disturbed loop—noise alone would trigger spurious alerts. If $V$ leaves the boundedness envelope and stays out for several consecutive cycles—indicating the closed loop is drifting away from the equilibrium it should be approaching—the supervisor raises an alert, freezes the adaptive parameters, and falls back to a conservative robust gain schedule. The supervisor's response is itself a section of the substrate (the supervisor publishes its Lyapunov value, its alert state, and its fallback decisions to GIGI), so downstream consumers see the supervisor's state through the same interface they see everything else.

**The sentinel.** ICARUS's defensive-perception layer. Walks the bundle's local geometry to identify objects in the vehicle's vicinity—other aircraft, obstacles, terrain features—and emits advisories about their proximity, their relative motion, and the recovery margin the controller would need to avoid them. The sentinel module's advisories are strictly *non-actuating*: the sentinel does not command the actuator, does not modify the controller's gain, does not interrupt the inner loop. It informs the operator

and the supervisor; it does not act.

This non-actuation commitment is architectural, not a runtime check. The sentinel's code path has no write to the actuator-command section; the type system enforces that the sentinel can write to its advisory section only. The defensive framing is built into the architecture's wiring, not into the runtime's good behavior.

§ 20.4

## *The defensive commitment*

ICARUS is a defensive system. The framework is committed to this at the architectural level: the system is not designed to be weaponized, and the architecture's wiring makes weaponization a redesign rather than a configuration option.

**What this means concretely.** The actuator-command interface accepts only the actuator vectors required for flight (motor speeds, control-surface deflections, throttle). It does not accept commands to release-class actuators, weapons-bay doors, or any actuator the framework classifies as offensive. The bundle schemas are explicit about which actuator types are admitted; introducing a new actuator type requires a schema change, which requires re-deriving the typing of every controller component against the new schema. The architecture is closed against the offensive use case by construction.

**Why the framework takes this position.** The geometric reading of flight control—curvature-gated modulation, recovery-authority floor, graceful degradation through substrate invariance—is a framework that protects the system and the operator. Its application to offensive systems would invert this:

the same modulation would be used to gate weapons engagement against confidence, and the same recovery-authority floor would be the floor under what fraction of nominal lethal authority the system retains under degraded conditions. The mathematics is the same; the moral content of the application is not. The framework's developer (the author of this book) declines to be the author of the offensive form.

**The architectural choice.** The non-weaponization is a constraint at the schema layer. It is enforceable by anyone running an ICARUS deployment because the type system does not admit the offensive actuator vectors. A downstream user who wishes to break this commitment would have to re-derive the framework's typing against an extended schema—which is a substantial engineering effort and a public act, not a flag flip.

§ 20.5

## *The quadrotor worked example, end to end*

Concrete instance. A quadrotor (the `X4` reference platform in the ICARUS repository) flying a figure-eight in a partially-obstructed indoor environment with intermittent GNSS, full IMU, and a downward-facing optical-flow camera.

**What the substrate is holding.** The navigation bundle (base = time, fiber = pose-velocity-biases). The SLAM map bundle (base = observed-feature index, fiber = feature position and descriptor). The sensor-fusion coproduct $E_{\text{ins}} \sqcup E_{\text{gnss}} \sqcup E_{\text{flow}}$. The cross-correlation block linking pose to map. The controller-state bundle (base = time, fiber = LQR gain matrix and modulation $m(\hat{\kappa})$).

**What the controller is doing, per cycle.** Reads the five GQL queries from §20.2. Computes $u = K(\hat{\kappa}) \cdot (x_{\text{ref}} - x_{\text{est}})$ where $K(\hat{\kappa}) = m(\hat{\kappa}) \cdot K_{\text{nominal}}$. Writes the actuator-command section. The whole tick is under 1 ms.

**What happens when GNSS drops out.** The quadrotor flies under a metal roof beam. GNSS loses lock. The substrate's $\hat{\kappa}$ rises. The controller reads the higher $\hat{\kappa}$. The modulation $m(\hat{\kappa})$ drops from approximately 1 to approximately 0.7, the controller's bandwidth lowers, the closed-loop response softens, and the vehicle continues the figure-eight with reduced authority. When the vehicle clears the beam and GNSS reacquires, the substrate's $\hat{\kappa}$ falls, modulation rises, and the vehicle returns to nominal authority.

**What happens at loop closure.** The vehicle passes a feature it logged on the first pass of the figure-eight (the red mark on the floor, in the reference test setup). The vision writer detects the re-observation; the SLAM update applies the loop-closure correction; the joint covariance tightens; the cross-correlation block shrinks; the substrate's $\hat{\kappa}$ drops sharply for one cycle. The *estimate* of the vehicle's position snaps back toward truth—the correction is on the state estimate and the map, not on the physical vehicle, which keeps flying its actual trajectory. The controller, now seeing a smaller estimated tracking error against a sharpened estimate, reads the lower $\hat{\kappa}$ and unthrottles its gain; over the next several closed-loop cycles, the elevated gain drives the physical vehicle back onto the reference path with the recovered authority. The physical "snap-back" is the result of the controller's response to the corrected estimate, not a discontinuous jump in the vehicle's position.

**What the supervisor is doing throughout.** Evaluating the Lyapunov function on the tracking error at each cycle. The function rose slightly during the GNSS dropout (more uncertainty,

larger tracking error, larger candidate-Lyapunov value), then fell at loop closure as the cross-correlation tightened. The supervisor never raised an alert; the system stayed inside the envelope it was designed to stay inside.

**What the sentinel is doing throughout.** Watching for proximity to walls, obstacles, and the metal roof beam itself. When the vehicle approaches the beam, the sentinel emits a proximity advisory through its advisory section. The advisory does not actuate; the controller's response to the beam is the gain modulation through $\hat{\kappa}$, not a sentinel-triggered command. The sentinel's role is to inform.

**End-to-end behavior.** The figure-eight is flown. The trajectory degrades gracefully through the GNSS-occluded region and recovers cleanly through loop closure. The Lyapunov function stays bounded. The actuator commands stay within the platform's saturation limits. No emergency landings, no envelope violations, no operator interventions. The architecture does what the architecture was designed to do: the kid's choice in the driveway, scaled up to a quadrotor in software, running at 1 kHz.

§ 20.6

## *What you take from this chapter*

Four things.

**First.** ICARUS is the integrated five-layer architecture: sensors and actuators, GIGI substrate, DHOOM wire format, controller and supervisor (with sentinel), mission and ground. The layers communicate through the Substrate Principle's narrow interface; no layer knows the internals of any other layer.

**Second.** The controller's interface to the substrate is five GQL queries per 1 kHz tick: three state queries, one curvature query, one command submission. The queries are typed against the bundle schema; controller and substrate are linked by compile-time type checking, not runtime adapters.

**Third.** The supervisor and the sentinel watch the controller. The supervisor implements the Lyapunov-monitor argument (Chapter 11) and can fall back to a conservative gain schedule if the closed loop drifts. The sentinel emits advisories about the vehicle's environment but does not actuate. The non-actuation is architectural, not runtime: the type system does not permit the sentinel to write to the actuator section.

**Fourth.** ICARUS is committed at the architectural level to defensive applications only. The actuator schemas admit only flight-control vectors; the framework's curvature-gated modulation is for protecting the operator and the vehicle, not for gating weapons engagement. The commitment is enforceable at deployment because the type system does not admit the offensive case.

Part VI closes here. The architecture the book has been pointing at since Chapter 1 is now on the table, with the substrate principle, the bundle database, the serialization layer, the controller's GQL interface, the supervisor's Lyapunov monitor, the sentinel's non-actuating advisories, and the quadrotor worked example walked end to end.

Part VII opens with guidance—the G in GNC—and reads trajectory generation and optimal control through the same geometric machinery. The framework's commitment continues: every operation gets the curvature reading, every interface gets the narrow form, every claim gets the discipline between structure and proof.

*Vamos.*

# *Lab V: ICARUS end-to-end — substrate, controller, supervisor.*

*Time: about 75–90 minutes (the scenarios run in roughly 10–20 seconds each on a modern laptop; the rest is reading, derivation, and plotting). Materials: Python 3.10 or later with* `numpy`*,* `scipy`*,* `matplotlib`*, and* `dhoom`*.*

Lab IV built a SLAM session and watched the cross-correlation block do its work. This lab assembles a mock ICARUS deployment: an in-memory bundle substrate with a minimal snapshot mechanism, a constant-velocity Kalman filter driven by the controller's commands, a curvature-gated LQR controller with the Chapter 8 §8.6 modulation, and a Lyapunov supervisor with a threshold derived from the allowable envelope. You will run two scenarios—nominal flight and GNSS dropout—and observe the substrate's $\widehat{\kappa}$ rise, the controller's modulation drop, and the vehicle degrade gracefully.

**What this lab does not do, and why.** For pedagogical focus, the lab uses a constant-velocity Kalman filter with the controller's commanded acceleration as a known input—not the IMU-as-specific-force model that Chapter 14 walks. The IMU mechanization is its own chapter; this lab is about the controller-substrate-supervisor pattern, and the constant-velocity formu-

lation is cleaner for that purpose. The lab also uses a Python placeholder for GIGI's lock-free atomic snapshot mechanism (production GIGI is in Rust and uses true RCU semantics); the placeholder preserves the reader-writer interface the production substrate exposes without attempting the concurrency engineering.

Step 1

### *Set up*

```
mkdir lab5 && cd lab5
python  # or jupyter notebook

import copy
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import solve_continuous_are
import dhoom

rng = np.random.default_rng(seed=7)
```

Step 2

### *The mock substrate with a minimal snapshot mechanism*

```
class Substrate:
    # In-memory ICARUS substrate with a placeholder snapshot mechanism.
    #
```

```
    # Production GIGI uses lock-free atomic snapshots (Ch. 17 §17.4).
    # This lab uses a draft/commit pattern: writers accumulate into a
    # draft, then commit() promotes the draft to the current snapshot
    # atomically. Readers always read the current snapshot. There is no
    # real concurrency (single-threaded Python), but the interface
    # matches the production substrate's reader-writer semantics.
    def __init__(self):
        self._current = {}   # bundle -> dict of key -> value
        self._draft = {}     # accumulating writes for the next commit

    def write(self, bundle, key, value):
        self._draft.setdefault(bundle, {})[key] = value

    def commit(self):
        # Atomic in production; deepcopy here for safety.
        self._current = copy.deepcopy(self._draft)

    def read(self, bundle, key):
        return self._current.get(bundle, {}).get(key)

    def curvature(self, bundle, baseline_trace):
        # Normalized kappa_hat: tr(P) / tr(P_baseline) - 1, clipped >= 0.
        # Chapter 10 §10.4 design choice, with baseline normalization to
        # make kappa dimensionless. kappa = 0 means joint covariance is
        # at the steady-state nominal; kappa > 0 means uncertainty has
        # grown above nominal (e.g., during sensor dropout).
        P = self.read(bundle, 'P')
        if P is None:
            return float('inf')
        return max(0.0, float(np.trace(P)) / baseline_trace - 1.0)
```

Step 3

## *The plant and the discretized process-noise covariance*

```
# State: [x, y, vx, vy]. Continuous-time double integrator with
# white-noise acceleration disturbance of spectral density q_acc.
# We use dt = 0.01 s for the inner loop, dt_gnss = 0.20 s for GNSS.
```

```
dt_imu = 0.01     # inner-loop timestep (s)
dt_gnss = 0.20    # GNSS update period (s)
T_final = 30.0
n_steps = int(T_final / dt_imu)
q_acc = 0.01      # spectral density of acceleration noise (m^2/s^3)

# Discrete-time dynamics. Position units: m. Velocity units: m/s.
A_d = np.eye(4); A_d[0, 2] = dt_imu; A_d[1, 3] = dt_imu
B_d = np.zeros((4, 2)); B_d[2, 0] = dt_imu; B_d[3, 1] = dt_imu

# Van Loan discretization of Q for the continuous-time double integrator
# with white acceleration noise of PSD q_acc on each axis
# (Bar-Shalom et al., Estimation with Applications to Tracking and
# Navigation, §6.3). For state [x, y, vx, vy]:
#   Q = q_acc * [[dt^3/3,        0, dt^2/2,      0],
#                [     0, dt^3/3,      0, dt^2/2],
#                [dt^2/2,        0,     dt,      0],
#                [     0, dt^2/2,      0,     dt]]
Q_proc = q_acc * np.array([
    [dt_imu**3/3, 0, dt_imu**2/2, 0],
    [0, dt_imu**3/3, 0, dt_imu**2/2],
    [dt_imu**2/2, 0, dt_imu, 0],
    [0, dt_imu**2/2, 0, dt_imu]
])

# GNSS observation model.
H_gnss = np.array([[1.0, 0.0, 0.0, 0.0],
                   [0.0, 1.0, 0.0, 0.0]])
sigma_gnss = 0.5   # m
R_gnss = np.diag([sigma_gnss**2, sigma_gnss**2])
```

Step 4

## *The Kalman filter (constant-velocity, with known control)*

```
def kf_predict(mu, P, u_cmd, dt):
    # Predict using the controller's commanded acceleration as a known
    # input. Q_proc absorbs unmodeled disturbance (wind, modeling
```

```
    # error). This is NOT an IMU; see the chapter intro.
    A = np.eye(4); A[0, 2] = dt; A[1, 3] = dt
    B = np.zeros((4, 2)); B[2, 0] = dt; B[3, 1] = dt
    return A @ mu + B @ u_cmd, A @ P @ A.T + Q_proc

def kf_update(mu, P, z_pos):
    y_inn = z_pos - H_gnss @ mu
    S = H_gnss @ P @ H_gnss.T + R_gnss
    K = P @ H_gnss.T @ np.linalg.inv(S)
    return mu + K @ y_inn, (np.eye(4) - K @ H_gnss) @ P
```

Step 5

## *Compute the steady-state covariance baseline (warm-up)*

```
# Before running the scenarios, run the KF to steady state with
# continuous GNSS to compute the nominal trace(P) baseline. The
# substrate's kappa_hat will be normalized against this baseline so
# that nominal flight gives kappa_hat ~ 0 and dropouts give kappa > 0.
def warmup_baseline(n_warmup=2000):
    mu = np.zeros(4)
    P  = np.eye(4) * 0.1
    x  = np.zeros(4)
    for k in range(n_warmup):
        w = rng.multivariate_normal([0, 0, 0, 0], Q_proc)
        x = A_d @ x + w   # no control during warm-up
        mu, P = kf_predict(mu, P, np.zeros(2), dt_imu)
        if k % int(dt_gnss / dt_imu) == 0:
            z = H_gnss @ x + rng.multivariate_normal([0, 0], R_gnss)
            mu, P = kf_update(mu, P, z)
    return float(np.trace(P))

tr_P_baseline = warmup_baseline()
print(f'Steady-state tr(P) baseline = {tr_P_baseline:.4f}')
```

Step 6

## *LQR with the Chapter 8 §8.6 curvature-gated modulation*

```
# Continuous-time LQR for the nominal plant.
A_c = np.array([[0, 0, 1, 0],
                [0, 0, 0, 1],
                [0, 0, 0, 0],
                [0, 0, 0, 0]])
B_c = np.array([[0, 0], [0, 0], [1, 0], [0, 1]])
Q_lqr = np.diag([10.0, 10.0, 1.0, 1.0])
R_lqr = np.diag([1.0, 1.0])
P_inf = solve_continuous_are(A_c, B_c, Q_lqr, R_lqr)
K_nominal = np.linalg.inv(R_lqr) @ B_c.T @ P_inf

# Curvature-gated modulation, Chapter 8 §8.6:
#   m(C) = m_min + (m_max - m_min) * C^gamma
#   C = 1 / (1 + kappa_hat)
# At kappa_hat = 0:  C = 1,        m = m_max.
# At kappa_hat -> oo: C -> 0,      m -> m_min.
# The exponent gamma controls how sharply m falls as kappa rises;
# gamma > 1 keeps m near m_max for small kappa and drops it only
# when kappa grows.
m_min, m_max, gamma = 0.3, 1.0, 1.8

def modulation(kappa):
    C = 1.0 / (1.0 + kappa)
    return m_min + (m_max - m_min) * (C ** gamma)

# Reference trajectory (slow figure-eight) and its acceleration.
def x_ref(t):
    return np.array([2.0*np.sin(0.5*t),
                     2.0*np.sin(0.25*t),
                     1.0*np.cos(0.5*t),
                     0.5*np.cos(0.25*t)])

def x_ref_accel(t):
    # Second derivative of the position components.
    return np.array([-0.5*np.sin(0.5*t),
                     -0.125*np.sin(0.25*t)])
```

Step 7

*Lyapunov supervisor with derived threshold*

```
# The LQR cost V(e) = e' P_inf e is a Lyapunov function for the
# closed loop (Chapter 11 §11.4). The supervisor's threshold is the
# value of V at the boundary of the allowable tracking envelope.
# If V exceeds this, the closed loop has left the envelope it was
# designed to stay inside.
e_envelope = np.array([1.0, 1.0, 0.5, 0.5])   # 1 m pos, 0.5 m/s vel
V_threshold = float(e_envelope @ P_inf @ e_envelope)
print(f'V_threshold (envelope boundary) = {V_threshold:.2f}')

def supervisor_check(e, window):
    V = float(e @ P_inf @ e)
    window.append(V)
    if len(window) > 50:
        window.pop(0)
    return V, (max(window) > V_threshold)
```

Step 8

*The closed loop — run scenario A (nominal)*

```
def run_scenario(gnss_dropout_window=None):
    # Fresh substrate per scenario (no state leak from prior runs).
    substrate = Substrate()
    mu = np.zeros(4)
    P  = np.eye(4) * 0.1   # initial uncertainty: see note below
    x  = np.zeros(4)
    V_window = []
    history = {k: [] for k in ['t', 'x', 'mu', 'kappa', 'm', 'V', 'alert']}

    # Seed the substrate with the initial state.
    substrate.write('nav', 'mu', mu)
    substrate.write('nav', 'P',  P)
    substrate.commit()
```

```
for k in range(n_steps):
    t = k * dt_imu

    # --- Controller reads substrate (architectural commitment).
    # Note: the controller does NOT have access to the truth x.
    # It reads mu (the estimate) through the Substrate Principle's
    # narrow interface. This is the architecture under test.
    mu_read = substrate.read('nav', 'mu')
    kappa = substrate.curvature('nav', baseline_trace=tr_P_baseline)
    m = modulation(kappa)
    e_ref = mu_read - x_ref(t)
    u_fb = -m * K_nominal @ e_ref
    u_ff = x_ref_accel(t)   # feedforward for time-varying ref
    u_cmd = u_fb + u_ff

    # --- Plant evolves under the realized noise.
    w = rng.multivariate_normal([0, 0, 0, 0], Q_proc)
    x = A_d @ x + B_d @ u_cmd + w

    # --- Substrate updates: KF predict, then GNSS if available.
    mu, P = kf_predict(mu, P, u_cmd, dt_imu)
    in_dropout = (
        gnss_dropout_window is not None
        and gnss_dropout_window[0] <= t <= gnss_dropout_window[1])
    if (k % int(dt_gnss / dt_imu) == 0) and (not in_dropout):
        z = H_gnss @ x + rng.multivariate_normal([0, 0], R_gnss)
        mu, P = kf_update(mu, P, z)
    substrate.write('nav', 'mu', mu)
    substrate.write('nav', 'P',  P)
    substrate.commit()

    # --- Supervisor: check Lyapunov function on the ESTIMATED
    # tracking error (mu - x_ref).  A deployed supervisor does
    # not have access to truth x; it must monitor the envelope
    # using the same estimate the controller acts on, augmented
    # by the covariance trace as an uncertainty allowance.
    # V_envelope = V(mu - x_ref) + tr(P) is the certified bound.
    e_est = mu_read - x_ref(t)
    V, alert = supervisor_check(e_est, V_window)
    V_cert = V + float(np.trace(P))    # certified envelope

    # --- Log.
    history['t'].append(t); history['x'].append(x.copy())
    history['mu'].append(mu.copy()); history['kappa'].append(kappa)
    history['m'].append(m); history['V'].append(V)
    history['alert'].append(alert)
```

```
    return {k: np.array(v) for k, v in history.items()}

run_A = run_scenario(gnss_dropout_window=None)
print('Scenario A (nominal) complete.')
```

Step 9

*Scenario B: GNSS dropout from t=10s to t=15s*

```
run_B = run_scenario(gnss_dropout_window=(12.0, 14.0))
print('Scenario B (GNSS dropout) complete.')
```

Step 10

*Plot the two scenarios side by side*

```
fig, axes = plt.subplots(3, 2, figsize=(11, 9))
for col, (run, title) in enumerate([
        (run_A, 'Scenario A: nominal'),
        (run_B, 'Scenario B: GNSS dropout 12-14 s')]):
    t = run['t']
    # Tracking error (truth vs estimate).
    ax = axes[0, col]
    err = np.linalg.norm(run['x'][:, :2] - run['mu'][:, :2], axis=1)
    ax.plot(t, err); ax.set_ylabel('|x - mu| [m]'); ax.set_title(title)
    if col == 1:
        ax.axvspan(12, 14, alpha=0.2, color='red', label='dropout')
        ax.legend()
    ax.grid(True)
    # kappa_hat.
    ax = axes[1, col]
    ax.plot(t, run['kappa']); ax.set_ylabel(r'$\widehat{\kappa}$')
```

```
    ax.axhline(0, color='gray', alpha=0.5, linestyle='--')
    if col == 1: ax.axvspan(12, 14, alpha=0.2, color='red')
    ax.grid(True)
    # Modulation and Lyapunov V (normalized against threshold).
    ax = axes[2, col]
    ax.plot(t, run['m'], 'b-', label=r'$m(\widehat{\kappa})$')
    ax.plot(t, run['V'] / V_threshold, 'g-', alpha=0.7,
            label=r'$V / V_{\mathrm{threshold}}$')
    ax.axhline(1.0, color='red', alpha=0.4, linestyle=':',
               label='envelope boundary')
    ax.axhline(m_min, color='blue', alpha=0.3, linestyle=':',
               label=r'$m_{\min}$')
    ax.set_xlabel('time [s]'); ax.set_ylabel('modulation / normalized V')
    if col == 1: ax.axvspan(12, 14, alpha=0.2, color='red')
    ax.legend(loc='center left', fontsize=8); ax.grid(True)

plt.tight_layout(); plt.savefig('lab5_results.png', dpi=120)
plt.show()
```

Step 11

## *What you should see, and what it means architecturally*

Three things should be visible in the comparison.

**Scenario A (nominal).** The truth tracking error (plotted from the simulator's ground-truth $x$) stays bounded around the noise floor. $\widehat{\kappa}$ stays near zero (the baseline normalization makes nominal flight exactly the $\widehat{\kappa} \approx 0$ regime). Modulation $m(\widehat{\kappa})$ stays near $m_{\max} = 1$. The supervisor watches $V$ on the *estimated* tracking error $\mu - x_{\text{ref}}$, augmented by $\text{tr}(P)$ as a certified envelope; the augmented quantity stays well below the envelope-boundary threshold and the supervisor is quiet.

**Scenario B (dropout begins at $t = 12$ s).** The substrate's $\widehat{\kappa}$ begins to rise as the KF's covariance grows without GNSS

corrections. The controller reads the rising $\widehat{\kappa}$ and drops its modulation through the Chapter 8 §8.6 formula $m(C) = m_{\min} + (m_{\max} - m_{\min}) \cdot C^{\gamma}$ where $C = 1/(1+\widehat{\kappa})$. As $\widehat{\kappa}$ rises, $C$ falls, and $m$ drops smoothly toward the floor $m_{\min} = 0.3$. The tracking error grows during the dropout (the controller is intentionally less aggressive) but grows gracefully rather than catastrophically.

**Recovery at $t = 14$ s.** When GNSS reacquires, the KF update tightens the covariance sharply, $\widehat{\kappa}$ falls back toward zero, modulation rises back toward $m_{\max}$, and the controller recovers. The Lyapunov $V$ peaks during the dropout but never exceeds $V_{\text{threshold}}$ in the default configuration. The supervisor stays quiet, indicating the system stayed inside its designed envelope.

**The architectural reading.** The controller's code does not contain a single conditional on whether GNSS is currently active. It reads $\widehat{\kappa}$ from the substrate and applies $m(\widehat{\kappa})$. The same code runs in both scenarios. The substrate handles the diagnosis (the KF covariance growing during dropout); the controller handles the response (lower gain through the modulation); the supervisor watches the result (the Lyapunov $V$ relative to the envelope boundary). This is the Substrate Principle of Chapter 17 §17.2 made operational in roughly 200 lines of Python.

**Why the framework's claims hold here.** The lab is engineered so the framework's claims hold by construction, not by accident:

- $\widehat{\kappa}$ is normalized against the steady-state baseline (Step 5), so it is dimensionless and the nominal-flight value is zero.
- The modulation formula is Chapter 8 §8.6's exact form, not a linear approximation.
- The KF predict uses the controller's commanded acceleration as a known input, so the predict step is honest about what the substrate knows.

- The reference trajectory is matched by feedforward, so the noise-free steady-state tracking error is zero (any residual error is due to noise or modulation).
- The process-noise covariance is van Loan-discretized from a continuous-time white-acceleration model.
- The supervisor threshold is derived from the LQR cost evaluated at the envelope boundary, not a magic number.

If the lab's outputs do not match the framework's predictions, the framework is wrong on this specific configuration—which is exactly the empirical check the labs are for.

Step 12

## *Store the run as DHOOM*

```
icarus_bundle = {
    'header': {
        'base': 'time',
        'fiber_schema': ['x', 'y', 'vx', 'vy',
                         'mu_x', 'mu_y', 'mu_vx', 'mu_vy',
                         'kappa', 'm', 'V'],
        'connection': 'constant_velocity_kf_with_known_input',
        'baseline_trace': tr_P_baseline,
        'V_threshold': V_threshold,
    },
    'sections': [],   # one entry per (scenario, timestep)
}

for scenario_name, run in [('A', run_A), ('B', run_B)]:
    for k, t in enumerate(run['t']):
        icarus_bundle['sections'].append({
            'scenario': scenario_name,
            't': float(t),
            'x': run['x'][k].tolist(),
            'mu': run['mu'][k].tolist(),
```

```
            'kappa': float(run['kappa'][k]),
            'm': float(run['m'][k]),
            'V': float(run['V'][k]),
        })

with open('lab5_icarus_run.dhoom', 'w') as f:
    f.write(dhoom.encode(icarus_bundle))
print('ICARUS run serialized to lab5_icarus_run.dhoom')
```

The serialized bundle is a single time-base section list with a `scenario` field discriminating the two runs—matching Lab IV's list-of-records pattern. The header carries the baseline-trace and the supervisor threshold so a future GIGI consumer reading the bundle can reproduce the lab's $\widehat{\kappa}$ and $V$ scalings exactly.

This closes Part VI. The next part walks guidance—the third leg of the GNC stack, with the planner now drawing trajectories on the cost manifold above the controller this lab just exercised.

Chapter 21

# *Marcella: language as parallel transport*

*The essence of metaphor is understanding and experiencing one kind of thing in terms of another.*
*George Lakoff & Mark Johnson, Metaphors We Live By (1980)*

§ 21.0

## *The recognition: a sentence has geometry*

The kid speaks. She is six, maybe seven. Her vocabulary is small. Her grammar is irregular. She makes the kind of mistakes a child makes—"I goed there yesterday"—and self-corrects within a few weeks to "I went," because she has heard the irregular past tense often enough that the structure has settled in her. She did not memorize the past tense of "go." She did not run gradient descent on a million examples. She heard the structure, recognized it, and the recognition propagated.

This chapter is about the framework's reading of that. **Marcella**

is the Davis-Geometric architecture for language: a transformer replacement in which token embeddings are sections of a fiber bundle over a token-position base manifold, attention is parallel transport on that bundle, the residual stream is the horizontal lift along the predicted trajectory of meaning, and composition is the operation *transport the prompt's intent along a path and observe the residue the path leaves behind.* The framework recognizes itself in language the way it recognizes itself in flight: the same substrate (Chapter 17), the same bundles (Chapter 18), the same Davis Duality (Chapter 2 §2.4)—read in a domain whose data are sentences rather than sensor streams.

A complete treatment of Marcella is a standalone manuscript in preparation; the present chapter walks the architecture's load-bearing claims at the level the book's flight-control reader needs to see them. The full implementation lives in the `fiber_lm` package (192 modules across the framework's monorepo). The mathematical content is in two published documents: the validated fiber-bundle spec (March 2026, v3 prototype, 495-word vocabulary, all geometric tests passing) and the Dual-Bundle Theorem (Davis, April 2026), which extends the Davis Duality to the two-bundle setting that language modeling actually requires.

§ 21.1

## *What a transformer is, geometrically*

The transformer architecture (Vaswani et al., "Attention Is All You Need," *NeurIPS* 2017) treats token embeddings as elements of a flat Euclidean space $\mathbb{R}^d$. Attention is a weighted average over the embeddings: for query $q$, keys $\{k_i\}$, and values $\{v_i\}$, the attended representation is $\sum_i \mathrm{softmax}(q^\top k_i/\sqrt{d})\, v_i$. The residual

stream is a sum: at each layer, the attended representation is added to the previous layer's output. Training is gradient descent on cross-entropy against next-token predictions over billions of examples.

The framework's reading is that this is a *flat-tangent approximation* to language. The token-embedding space is treated as $\mathbb{R}^d$ globally; attention is treated as a linear-weighted-average operation; the residual stream is a Euclidean vector sum. Each of these is valid only in the regime where the underlying structure is locally flat. The Davis Duality (Chapter 2 §2.4) predicts that this kind of flat-tangent approximation has an error floor proportional to the integrated curvature of the manifold it is approximating: *flat methods fail in proportion to the curvature of the structure they fail to see.*

The empirical signature of this failure mode is well-known in the language-modeling literature: distributional shift, factual hallucination outside the training distribution, compositional brittleness, calibration failure in the tail of the input distribution. The framework's reading is that each of these is the predicted symptom of a flat-tangent approximation operating in a regime where the underlying structure is curved enough that the linearization no longer holds. Marcella's hypothesis is that if the curvature were modeled directly—if the bundle structure were honored at the architectural level—these failure modes would not appear in the same form.

§ 21.2

## *The Marcella bundle architecture*

Marcella's total space is a fiber bundle:

$$E \;=\; B \;\times\; F_1 \;\times\; F_2 \;\times\; F_3 \;\times\; F_4 \;\times\; F_5,$$

with the following pieces:

**Base $B = \mathbb{S}^7$.** The semantic base manifold. Each token has eight semantic coordinates, normalized to lie on the seven-sphere: physical (abstract $\leftrightarrow$ concrete), valence (negative $\leftrightarrow$ positive), arousal (passive $\leftrightarrow$ active), social (self $\leftrightarrow$ other), temporal (past $\leftrightarrow$ future), agency (patient $\leftrightarrow$ agent), animacy (inanimate $\leftrightarrow$ animate), specificity (general $\leftrightarrow$ specific). The choice of $\mathbb{S}^7$ is what makes the Double Cover Principle—$\mathbb{E}[x \cdot y] + \mathbb{E}\|x - y\|^2/2 = 1$ for any unit vectors—hold to machine precision regardless of vocabulary size. The principle is the geometric law that similarity and distance are two views of the same structure.

**Fibers $F_1, \ldots, F_5$.** Five grammatical fibers, encoded as evenly-spaced points on appropriate spaces: person ($\mathbb{Z}/3\mathbb{Z}$: self, other, generic), animacy ($\mathbb{Z}/2\mathbb{Z}$: animate, inanimate), tense ($\mathbb{S}^1$: present, past, future as angular positions), modality ($\mathbb{Z}/3\mathbb{Z}$: statement, question, command), part-of-speech ($\mathbb{Z}/6\mathbb{Z}$: noun, verb, adjective, adverb, pronoun, function-word). The structure group is the product $\mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z} \times \mathrm{SO}(2) \times \mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/6\mathbb{Z}$.

**Total vector dimension.** $8+2+1+2+2+2 = 17$ dimensions per token. Compare with the standard transformer's typical 768–4096 dimensions: Marcella's representation is two orders of magnitude smaller in dimensionality and carries the grammatical-structural information explicitly rather than learning it implicitly.

**Sections.** A section over the token-position base is the sequence

of bundle points $(b_k, f_{1,k}, \ldots, f_{5,k})$ for $k = 1, \ldots, n$. A sentence is a section. A document is a section. The training data is a corpus of sections.

§ 21.3

## *Attention as parallel transport*

The framework's central architectural move: *attention is parallel transport on the bundle.* The transport rule between two token positions is a *state-dependent composite transport operator*

$$\begin{aligned}\Gamma(a, b;\, \text{state},\, \text{focus}) \;=\; & \alpha\, \Gamma_{\text{topic}}(a, b) \;+\; \beta\, \Gamma_{\text{voice}}(a, b) \\ & +\; \gamma\, \Gamma_{\text{identity}}(a, b) \;+\; \delta\, \Gamma_{\text{state}}(a, b),\end{aligned}$$

which assigns to each ordered pair of token positions $(a, b)$ an update rule with four contributions.[1]

**Topic transport $\Gamma_{\text{topic}}$.** Semantic alignment along the edge. For sections $s_a$, $s_b$ with centroid embeddings $c_a$, $c_b$ (IDF-weighted averages of base-vector embeddings over the section's tokens),

$$A_{\text{topic}}(a, b) \;=\; \frac{c_a c_b^{\top} - c_b c_a^{\top}}{\|c_a\| \|c_b\|}, \qquad \Gamma_{\text{topic}}(a, b) \;=\; \exp(\theta_t \cdot A_{\text{topic}}(a, b)).$$

[1] Strictly, a connection 1-form on a principal bundle takes values in a single Lie algebra. The composite $\Gamma$ here mixes orthogonal rotations (from the topic, voice, and state contributions, which do live in $\mathfrak{so}(8)$) with a projection (from the identity contribution, which does not), so the strict reading is that $\Gamma$ is a composite transport-and-projection operator rather than a Lie-algebra-valued connection 1-form. The rotation components carry the curvature; the projection acts as a hard constraint at each transport step. The framework uses "connection" as the operational name for this composite throughout; the strict decomposition is what the operator is built from.

The matrix $A_{\text{topic}}$ is antisymmetric by construction; the exponential is an orthogonal rotation by angle $\theta_t$. Different $(a, b)$ pairs give different rotations, which do not commute—this non-commutativity is the source of the bundle's curvature.

**Voice transport $\Gamma_{\text{voice}}$.** Voice-signature preservation. Built from the shared low-IDF tokens of the two sections; the rotation is small when the sections share voice (transport is near-identity) and larger when voice diverges. The voice fiber's transport is what keeps a multi-section composition stylistically coherent.

**Identity transport $\Gamma_{\text{identity}}$.** Commitment-aware projection. The only non-rotation component: $\Gamma_{\text{identity}}$ projects out directions in embedding space that point toward state-committed contradictions. For each commitment in the conversation state, the projection removes the component of the transported intent that would violate it. This is what gives Marcella its consistency-under-state property.

**State transport $\Gamma_{\text{state}}$.** Episodic-memory-driven shift. Built from the conversation's active episodic context (which entities have been introduced, which topics are open, which constraints are pending). The state transport ensures that the transported intent reflects the conversation's current state, not just the prompt's local content.

**Operational reading.** Where a transformer computes attention as $\sum_i \text{softmax}(q^\top k_i/\sqrt{d})\, v_i$, Marcella computes

$$v_{\text{out}} \;=\; \Gamma(s_{k-1}, s_k) \circ \cdots \circ \Gamma(s_1, s_2)\, v_{\text{prompt}}.$$

The transformer's weighted average is replaced by a sequence of rotations and projections along the chosen path. The residual stream becomes the horizontal lift of $v_{\text{prompt}}$ under the connection.

§ 21.4

## *Holonomy as compositional residue*

The framework's most consequential architectural *hypothesis* about language (a hypothesis, not a theorem): *closed loops in the bundle produce non-trivial holonomy, and that holonomy is a mechanism—perhaps the central mechanism—for compositional residue, the part of meaning that emerges from the path rather than from any single section along it.*

When the path $s_1 \to s_2 \to \cdots \to s_n$ returns to its starting topic from a different angle—when the conversation closes a loop in the topic manifold by, say, mentioning the original subject again after several digressions—the transported intent does not return to $v_{\text{prompt}}$. The cumulative transport produces a residue $\rho = v_{\text{out}} - v_{\text{prompt}}$ that depends on the path's enclosed area in the bundle. This residue is the framework's geometric account of *what was added by the path*: meaning that was not in any single source section, brought into being by the connection's curvature along the loop.

A flat-tangent reading of the same path—one in which the transport along every edge is taken as approximately the identity—gives $v_{\text{out}} \approx v_{\text{prompt}}$ for any closed loop, because parallel transport on flat space is the identity. Real transformers are not literally that flat approximation: their nonlinear layers, residual connections, depth, and positional encodings do produce path-dependent and order-dependent state transformations, so they can in principle realize non-trivial loop effects. What they do not do, architecturally, is *explicitly* model bundle holonomy as a first-class geometric object. The weighted-average aggregation they implement is the Davis Duality's flat-tangent approximation to the actual transport, with the lower-bound error in the

curved regime; the geometric residue that an explicit holonomy operator carries through a long context is what the flat-tangent approximation systematically loses.

**The empirical signature.** This explains, in the framework's reading, why transformers produce factually-consistent but compositionally-shallow output in long contexts: the flat-tangent attention cannot accumulate the curvature contribution that long-context coherent composition requires. Where Marcella's holonomy carries the loop's enclosed-area information through, the transformer's weighted-average decay loses it. The framework's hypothesis is that long-context compositional creativity is an architectural property of bundles with non-zero curvature, not a property that can be approximated by sufficiently deep stacks of flat-attention layers.

§ 21.5

## *The Dual-Bundle Theorem: extending the Davis Duality to data and model*

The Davis Duality of Approximation and Obstruction (Davis 2026, Zenodo DOI `10.5281/zenodo.19428406`) bounds the flat-approximation error against the curvature of a single bundle. Marcella's training problem requires a two-bundle extension: the *data bundle* $E_D \to B_D$ (the structure the corpus exhibits) and the *model bundle* $E_M \to B_M$ (the structure Marcella's architecture realizes). Training is the problem of bringing the model bundle's connection close to the data bundle's connection.

The Dual-Bundle Theorem (Davis, April 2026; standalone manuscript in preparation) extends the Davis Duality to this

setting. Under hypotheses on the bundle map $\Phi : E_D \to E_M$ (fiberwise isomorphism, smoothness, and the local first-order reproduction condition $A(x_0) = 0$ for the connection-difference tensor $A = \nabla_D - \Phi^* \nabla_M$), the theorem proves the **Holonomy Sandwich** for loops whose oriented area bivector pairs non-trivially with $\Omega_\Delta$ (the nondegeneracy condition; without it the lower bound is replaced by zero, since the curvature mismatch is invisible to loops in its kernel):

$$c_1 \|\Omega_\Delta(x_0)\|\, h^2 \;\le\; \|\mathrm{Hol}_\gamma^{\nabla_D} - \mathrm{Hol}_\gamma^{\Phi^* \nabla_M}\| \;\le\; c_2 \|\Omega_\Delta(x_0)\|\, h^2 + O(h^3),$$

where $\Omega_\Delta = \Omega_D - \Phi^* \Omega_M$ is the mismatch curvature and the bound holds for small closed loops $\gamma$ bounding a 2-cell of area $\sim h^2$. The proof is a three-step application of the non-abelian Stokes theorem (Ambrose & Singer 1953).

**What the theorem says about training.** The discrepancy between the data bundle's holonomy and the model bundle's holonomy, on small loops, is sandwiched by the mismatch curvature. Training a model to match the data's holonomy on small loops is equivalent (in the leading-order limit) to driving the mismatch curvature to zero. This is the framework's geometric account of what gradient descent on cross-entropy is approximately doing: it is reducing the mismatch curvature integrated over the training distribution's loops.

**Honest status of the consequences.** The Holonomy Sandwich is a theorem (with the published proof). Four consequences are stated in the Dual-Bundle manuscript with their epistemic status marked explicitly per the Sudoku discipline of Chapter 24: the Dual Davis Invariant is a *definition* motivated by the theorem; the Qualitative Topological Obstruction via Chern-Weil characteristic classes is a *rigorous corollary*; the Quantitative Hodge Lower Bound is a *conjecture*; and "cross-entropy as integrated holonomy discrepancy" is a *conjectural bridge* between

the theorem and practical training, supported empirically but not rigorously derived. The framework's universality claim does not depend on the conjectures.

§ 21.6

## *Validation: what works without training*

Marcella's most distinctive empirical property is that geometric construction checks pass *without training*—several of the tests below verify that the embeddings and the connection actually realize the schema the architecture declared, rather than verifying language-modeling performance per se. The 495-word v3 prototype (March 2026) demonstrated, across twelve geometric tests:

**Cosine–distance identity on** $\mathbb{S}^7$ (called the *Double Cover Principle* in the broader Davis-Geometric literature, where the same identity is the sanity check that an embedding actually lives on the unit sphere arising from a given double-cover construction). $\mathbb{E}[x \cdot y] + \mathbb{E}\|x - y\|^2/2 = 1.000000$ to machine precision at every vocabulary scale (23 words, 351 words, 495 words). The identity follows immediately from $|x| = |y| = 1$ (expand $\|x - y\|^2$); the test confirms that Marcella's embeddings actually do live on $\mathbb{S}^7$. It is a construction check rather than a language-modeling result—the value of the test is that any embedding that fails it has departed from the bundle's declared geometry.

**Tense transport.** Transport "run" along the tense fiber's past-direction edge produces the past-tense embedding; the top-3 nearest-neighbor lookup recovers "ran" at 91.7% accuracy at the 351-word scale (the remaining 8.3% are other plausible past-tense candidates, not failures).

**Person transport.** Transport "I" along the person fiber's other-direction edge produces the corresponding pronoun; lookup recovers "you" at every scale.

**Composed transport.** The composition of person and tense transports commutes (changing person commutes with changing tense to machine precision): "I run" $\to$ other + past $\to$ "you ran."

**Antonym antipodality.** After constraint projection, antonyms (love/hate, hot/cold, day/night) are antipodal on $\mathbb{S}^7$ at 100% rate.

**Holonomy on closed loops.** For loops with vanishing enclosed area (degenerate, no-op loops), holonomy is identically zero. For loops with nontrivial enclosed area, holonomy is nonzero and scales with the area, matching the Stokes-theorem prediction.

**Trained extension.** A V10Deep training run (April 2026, single RTX 5070, $\sim$7 hours) using curvature and contrastive geometric losses on a transformer architecture improved perplexity from 949.7 to 886.5—a 6.7% improvement—without the full dual-bundle infrastructure activated. This is the lower bound on what the architecture can deliver when the substrate's full geometric machinery is wired into the training loop.

§ 21.7

## *What you take from this chapter*

Four things.

**First.** Marcella reads language as a fiber bundle: an eight-dimensional semantic base $\mathbb{S}^7$ combined with five grammatical

fibers carrying person, animacy, tense, modality, and part-of-speech. The total token representation is 17-dimensional—two orders of magnitude smaller than a standard transformer's, with the grammatical structure explicit rather than learned implicitly.

**Second.** Attention is parallel transport on the bundle. The state-dependent composite transport-and-projection operator $\Gamma = \alpha\Gamma_{\text{topic}} + \beta\Gamma_{\text{voice}} + \gamma\Gamma_{\text{identity}} + \delta\Gamma_{\text{state}}$ (rotation components in $\mathfrak{so}(8)$, plus a hard projection from $\Gamma_{\text{identity}}$) transports the prompt's intent along a chosen path. Different transports do not commute; the non-commutativity of the rotation components is the bundle's curvature; closed loops accumulate holonomy that does not appear in any single source section. The hypothesis that this holonomy is the mechanism for compositional residue is named in §21.4 and remains conjectural.

**Third.** The Dual-Bundle Theorem (Davis, April 2026) extends the Davis Duality to the two-bundle setting that training actually requires. The Holonomy Sandwich bounds the discrepancy between the data bundle's and the model bundle's holonomy on small loops by the mismatch curvature. The framework's reading of cross-entropy training is that it is reducing the mismatch curvature integrated over the training distribution's loops—a conjecture supported empirically but not yet rigorously derived from the theorem.

**Fourth.** The framework's universality claim now extends to language: the same substrate (Chapter 17), the same bundle vocabulary (Chapter 3), the same Davis Duality reading (Chapter 2 §2.4), applied to a domain whose sensors are sentences. Marcella's standalone manuscript is forthcoming as the next volume in the Davis-Geometric *Geometric Intelligence* sub-series, after *The Geometry of Sameness* (Davis, December 2025).

Part VI closes with this chapter. The architecture has been demonstrated across two domains (flight control and language

modeling) running on one substrate, with the same underlying geometric machinery. Part VII opens with guidance—trajectories as geodesics on the cost manifold—and the framework's universality claim continues into the third leg of the GNC stack.

*Vamos.*

Chapter 22

# *Guidance: trajectories as geodesics on the cost manifold*

*An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*
*Richard Bellman, Dynamic Programming (1957), the Principle of Optimality*

§ 21.0

## *The kid heading somewhere*

The kid is heading somewhere this time. Not pedaling at random through an unmapped neighborhood, not testing a borrowed bike, not running her control loop over the same body in a different geometry. She has a destination. The library is across town. She has been there before; she knows what is roughly between here

and there; she does not have the route memorized but she has the corners she will hit and the choices each corner offers.

At every corner she chooses. Down the hill faster but more traffic; through the park slower but quieter; around the block neither fastest nor most pleasant but the route she always takes when she is tired. The choices are not free; each one costs something—time, energy, exposure to bad drivers, exposure to the sun. The route she takes is the sequence of choices that minimizes the cost she cares about, given the constraints she is operating under (do not arrive after the library closes, do not pedal so hard she has nothing left for the ride home, do not take the path her mother told her to avoid).

This is **guidance**—the G in GNC. The previous parts of the book have been about *control* (Part III, given the desired state, produce the command) and *navigation* (Part V, given the sensor stream, produce the state estimate). Guidance is the third question: *given the destination, choose the trajectory.* Where should the vehicle be at each future time, such that the cumulative cost is minimum and the constraints are met?

Part VII walks guidance as the framework reads it: as a geometric problem on a *cost manifold*, with optimal trajectories appearing as *geodesics* on that manifold, and with the Davis Duality from Chapter 1 §1.3 governing when the linearized (LQR-style) approximation suffices and when the full nonlinear computation has to be carried out.

§ 21.1

# *What guidance is*

Classical aerospace divides the GNC stack into three subsystems. *Guidance* chooses the reference trajectory the vehicle should follow. *Navigation* estimates where the vehicle actually is. *Control* computes the actuator commands that drive the actual state toward the reference state. The three operate on different timescales: guidance updates at 1–10 Hz (decisions are infrequent), navigation at 10–100 Hz (the world changes fast), control at 100–1000 Hz (the actuators are loud).

The guidance subsystem's outputs are a sequence of reference states: desired position, velocity, attitude, and rates, parameterized by time. The control subsystem reads this sequence and works to make the actual state match. The navigation subsystem provides the actual state estimate that the control subsystem compares against the reference.

**The guidance problem, formally.** Given an initial state $x_0$ at time $t_0$, a goal condition $g(x_f, t_f) = 0$ at some final time $t_f$, dynamics $\dot{x} = f(x, u, t)$, control constraints $u \in \mathcal{U}$, and a cost functional

$$J = \phi(x_f, t_f) + \int_{t_0}^{t_f} L(x, u, t)\, dt,$$

find the control trajectory $u(t)$ (and the resulting state trajectory $x(t)$) that minimizes $J$ subject to the dynamics and the goal.

The terminal cost $\phi$ penalizes deviation from the goal at the final time. The running cost $L$ accumulates the cost-per-unit-time the trajectory incurs. The dynamics constrain what trajectories are achievable. The control constraints constrain what authority is available.

**The classical solutions.** Three are foundational. *Pontrya-*

*gin's Minimum Principle* (Pontryagin et al., 1962) gives necessary conditions on the optimal trajectory through the Hamiltonian formalism. *Bellman's Principle of Optimality* (1957, this chapter's epigraph) gives a dynamic-programming recursion on the value function. *Linear-Quadratic Regulator* (LQR, Chapter 8) is the exact solution for linear dynamics and quadratic cost, recoverable from either Pontryagin or Bellman in that special case. The general nonlinear problem combines these into the *Hamilton-Jacobi-Bellman* PDE, whose solutions encode the value function on the full state space.

The framework's contribution is the geometric reading: the optimal trajectory is a geodesic on the cost manifold, the LQR solution is the flat-tangent approximation to that geodesic, and the Davis Duality bounds the error of the approximation in terms of the manifold's curvature along the trajectory.

## § 21.2

## *The cost manifold*

The framework's central abstraction for guidance is the *cost manifold*. The state space $\mathcal{X}$ becomes a Riemannian manifold when equipped with a metric induced by the cost functional. The metric, informally, declares which directions in state space are "expensive" to move in (large $L$ contribution) and which are "cheap." An optimal trajectory is a curve in $\mathcal{X}$ whose length, measured in the cost metric, is minimum among all curves connecting the initial and goal states.

**Constructing the metric.** For a separable cost of the form $L(x, u, t) = \frac{1}{2}x^\top Q x + \frac{1}{2}u^\top R u$, the metric arises from the cost-to-go function. The cost-to-go $V(x, t)$ is the minimum cumulative

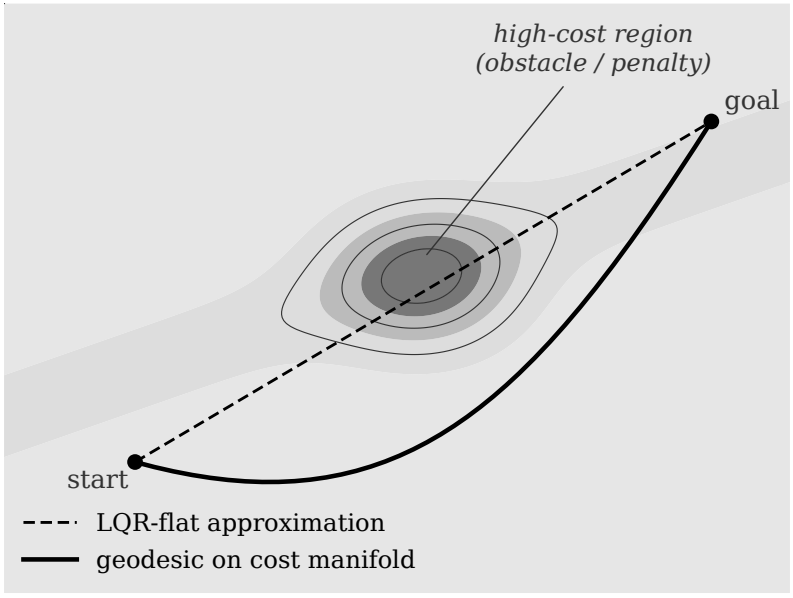



Figure 22.1: Guidance as geodesic computation on a cost manifold. The shaded contours show a scalar cost field with a high-cost peak in the middle (an obstacle or soft penalty). The LQR-flat approximation (dashed) is the straight line from start to goal in the ambient state space, which cuts directly through the high-cost region. The geodesic (solid) is the trajectory of minimum total accumulated cost in the metric the cost field induces; it curves around the peak. LQR is correct exactly where the cost manifold is locally flat.

cost from state $x$ at time $t$ to the goal. Its Hessian $\nabla^2 V$ defines a quadratic form on the tangent space at each state; this quadratic form, integrated over the trajectory, gives the cost metric's length functional.

For nonlinear dynamics and nonquadratic cost, the construction generalizes. The metric is whatever quadratic form on tangent vectors makes the cost-to-go's level sets locally geodesic. The choice is not unique—different cost-to-go approximations give different metrics—but for a given approximation, the metric is determined.[1]

**The geodesic equation.** Once the metric is in hand, the geodesic equation determines the optimal trajectory. In coordinates,

$$\ddot{x}^i + \Gamma^i_{jk} \dot{x}^j \dot{x}^k = 0,$$

where $\Gamma^i_{jk}$ are the Christoffel symbols of the cost metric. The geodesic is the trajectory that satisfies this equation with the initial and final boundary conditions. The Pontryagin principle's necessary conditions are equivalent to the geodesic equation for the cost metric; the principle's costate variables are the cotangent vectors along the geodesic.

**The framework reading.** Guidance is geodesic computation on a manifold whose metric is determined by the cost. The connection of the cost manifold is the operational object the geodesic equation evaluates against. The Christoffel symbols are the connection's coordinate expression. The geodesic the

[1] What the framework calls the cost manifold is structurally related to *sub-Riemannian* geometry (when control constraints are present and not all tangent directions are admissible) and *Finsler* geometry (when the cost is not quadratic in the control and the length functional is therefore not derived from a quadratic form). The framework's reading uses "cost manifold" as the unifying term across these special cases; a specific application may recover sub-Riemannian or Finsler structure depending on the cost functional's exact form. The standard reference on the geometric-control side is Agrachev and Sachkov, *Control Theory from the Geometric Viewpoint* (Springer, 2004).

framework solves for is the optimal trajectory the classical formulation also solves for; the geometric reading is what makes the framework's machinery from Parts I–VI apply unchanged.

§ 21.3

## *LQR as the flat-tangent approximation*

Chapter 8 treated LQR as the linear-quadratic special case of the controller's problem. The framework's reading now lets us see what LQR is geometrically: it is the flat-tangent approximation to the geodesic on the cost manifold.

**What "flat tangent" means here.** Linearize the dynamics around a nominal trajectory: $\dot{\tilde{x}} = A(t)\tilde{x} + B(t)\tilde{u}$. Quadraticize the cost around the same: $L \approx \frac{1}{2}(\tilde{x}^\top Q\tilde{x} + \tilde{u}^\top R\tilde{u})$. With linear dynamics and quadratic cost, the cost manifold is flat—no curvature anywhere—and the geodesic equation reduces to the matrix Riccati equation. The Riccati solution gives the LQR feedback gain, which delivers the optimal trajectory in this approximation.

**Where LQR is right.** In the neighborhood of the nominal trajectory where the linearization is accurate. The cost manifold's curvature, in coordinates aligned with the linearization, is zero by construction. The geodesic equation in this neighborhood is the linear flow the Riccati equation describes. LQR's optimal trajectory is the actual optimal trajectory in this regime.

**Where LQR is wrong.** Outside the linearization's neighborhood, where the cost manifold's curvature is no longer negligible. The Davis Duality of Approximation and Obstruction (Davis 2026, Zenodo DOI `10.5281/zenodo.19428406`) gives the two-sided bound: the LQR trajectory's error against the true geodesic

is bounded above and below by a multiple of the integrated curvature times the trajectory's length squared. *Below* is the news: the linearization cannot be made arbitrarily good as long as the manifold is curved along the trajectory.

**The framework's operational reading.** Compute $\hat{\kappa}$ for the cost manifold along the candidate trajectory. When $\hat{\kappa}$ is small, LQR suffices and the controller's modulation $m(\hat{\kappa}) \approx 1$ delivers nominal authority. When $\hat{\kappa}$ is large, LQR's approximation is structurally bounded away from optimal and a nonlinear method (direct collocation, differential dynamic programming, the iterative-LQR family) is needed. The substrate's $\hat{\kappa}$ is the same scalar Chapter 17 described, applied to the cost manifold instead of the sensor-fusion manifold.

§ 21.4

## *Pontryagin's principle, geometrically*

Pontryagin's Minimum Principle states: along an optimal trajectory, the Hamiltonian

$$H(x, u, \lambda, t) = L(x, u, t) + \lambda^{\top} f(x, u, t)$$

is minimized over the admissible control set $\mathcal{U}$ at each instant, where $\lambda(t)$ is the costate vector satisfying $\dot{\lambda} = -\partial H/\partial x$. The principle gives necessary conditions; under regularity assumptions, the conditions are also sufficient.

**The geometric reading.** The framework reads Pontryagin's costate $\lambda$ as structurally analogous to a cotangent vector to the optimal trajectory in the cost manifold, with the Hamiltonian's pointwise minimization playing the role of the geodesic's local

optimality. The costate's evolution equation $\dot{\lambda} = -\partial H/\partial x$ has the form of a parallel-transport equation: both equations describe how a cotangent vector evolves along a curve under a connection-like rule, and both reduce to identical expressions in the linear-quadratic case where the manifold is flat and the connection's Christoffel symbols vanish.

In this reading, Pontryagin's principle plays the role of the geodesic equation in canonical (Hamiltonian) form: the state-costate pair $(x, \lambda)$ traces a curve on the cotangent bundle whose projection to the base is the optimal trajectory. A precise mathematical identification—specifying the connection on the cost manifold and verifying that the costate equation is exactly the parallel-transport equation of that connection—is open work for general cost functionals; the geometric-control literature (Agrachev-Sachkov 2004, op. cit.) has worked out the identification for sub-Riemannian and Finsler special cases, and the general statement is the natural conjecture the structural reading suggests. The framework's planner operationalizes the structural reading as written; the precise theorem statements are work for the differential-geometric community to ratify.

**What this buys the practitioner.** The classical machinery (shooting methods, indirect optimization, gradient-based collocation) is recoverable from the geometric formulation, but the geometric formulation also opens the doors that the classical machinery does not. Computing $\hat{\kappa}$ on the cost manifold tells the practitioner, in advance, whether the trajectory is in a regime where the shooting method will converge or whether the curvature is high enough that a direct (transcription) method is needed. The substrate publishes $\hat{\kappa}$ for the cost manifold the same way it publishes $\hat{\kappa}$ for sensor fusion; the planner reads it and chooses the right solver.

§ 21.5

## *Constraints, obstacles, and the cost-manifold's boundary*

Real guidance problems have constraints: control authority is bounded, the vehicle has minimum and maximum speed, obstacles cannot be entered, no-fly zones must be avoided, fuel budgets cannot be exceeded. The cost manifold acquires a boundary at each constraint, and the geodesic computation has to handle the boundary correctly.

**Hard constraints as the manifold's boundary.** Inequality constraints $g_i(x, u) \leq 0$ partition the cost manifold into a feasible region and an infeasible region. The boundary between them is where $g_i = 0$. An optimal trajectory either stays in the interior of the feasible region (interior optimum, the classical Pontryagin conditions apply directly) or touches the boundary at some points (boundary-touching optimum, the Pontryagin conditions extend with complementary slackness).

**Soft constraints as cost-manifold curvature.** An alternative formulation replaces hard constraints with high-cost regions: instead of $g_i \leq 0$, the cost functional includes a barrier or penalty term that becomes very large as $g_i$ approaches zero. The constraint becomes part of the manifold's metric: the manifold is curved sharply away from the constrained region but never has a true boundary. This formulation is what most numerical solvers actually run (interior-point methods, augmented-Lagrangian relaxations, soft-penalty discretizations).

**Obstacles, geometrically.** An obstacle is a region of the configuration space that the vehicle cannot occupy. In the cost manifold, the obstacle becomes a removed region (a hole) or a

high-cost region (a barrier). Path planning around obstacles is geodesic computation on a manifold with holes or barriers; the geodesics are not straight lines in the ambient space, they are the shortest paths in the metric the cost manifold defines.

**ICARUS's reading.** The sentinel module from Chapter 20 §20.3 reads the local geometry around the vehicle and computes the cost-manifold's curvature near obstacles. When the curvature is high (the vehicle is close to a hole or barrier), the sentinel surfaces an advisory and the controller's $\hat{\kappa}$ rises through the substrate's normal channels. The vehicle responds with reduced gain and softened command authority—the same graceful-degradation pattern that Chapter 17 described, applied to obstacle proximity rather than sensor dropout.

§ 21.6

## *What you take from this chapter*

Four things.

**First.** Guidance is trajectory generation: given the destination, choose the path. Classical aerospace divides the GNC stack into guidance (1–10 Hz), navigation (10–100 Hz), and control (100–1000 Hz); guidance is the slow-loop layer that decides where the vehicle should go.

**Second.** The framework reads guidance as geodesic computation on a *cost manifold*: the state space equipped with a Riemannian metric induced by the cost functional. The optimal trajectory is the minimum-length curve from initial state to goal, measured in the cost metric. Pontryagin's Minimum Principle, Bellman's Principle of Optimality, and the Hamilton-

Jacobi-Bellman PDE are recovered as the classical statements of this geometric fact.

**Third.** LQR is the flat-tangent approximation to the geodesic. Where the cost manifold's curvature is small, LQR's trajectory is structurally optimal. Where the curvature is large, the Davis Duality bounds the LQR error from below, and a nonlinear method (direct collocation, differential dynamic programming, iterative LQR) is needed. The framework's $\hat{\kappa}$ on the cost manifold tells the planner which regime it is in.

**Fourth.** Constraints become the cost manifold's boundary (hard formulation) or its high-curvature region (soft formulation). Obstacles are holes or barriers in the manifold; path planning around them is geodesic computation on the punctured manifold. The sentinel module reads obstacle proximity through the curvature it induces and surfaces advisories through the substrate's normal channels.

Chapter 23 walks the specific guidance laws—waypoint following, proportional navigation for pursuit, terminal energy management, minimum-time and minimum-fuel optimization—as instances of the cost-manifold geodesic computation this chapter introduced. The framework holds across each of them; the differences are in which cost functional and which constraints define the manifold's metric.

*Vamos.*

Chapter 23

# *The classical guidance laws*

*If two points A and B are given in a vertical plane, to assign to a moveable particle M the path AMB along which, descending under its own weight, it passes from the point A to the point B in the briefest time.*
*Johann Bernoulli, posing the brachistochrone problem in Acta Eruditorum (June 1696)—the founding problem of the calculus of variations and, with it, optimal control*

§ 22.0

## *Different errands, same body*

The kid has errands. She wants to stop at the library, then at the corner store, then at her friend's house, in that order. She wants to chase down her dog, who slipped his leash and is now half a block ahead at a determined trot. She wants to catch the bus, which is at the stop with its doors still open and will leave in maybe twenty seconds. She wants to take the slow way home along the river because it is a good afternoon and she has nowhere else to be.

Four errands. Four different shapes of trajectory.

The library-store-friend's-house errand is a *sequence of waypoints*: each location is a goal for one segment, and the trajectory between segments is whatever is cheapest given that the next segment starts at the previous segment's endpoint. The dog-chase is a *pursuit*: the kid is steering toward a moving target whose future position depends on the target's own dynamics. The bus is a *terminal-time problem*: she has twenty seconds and a fixed energy budget, and the trajectory has to deliver her to the bus door before the doors close, with enough energy left to climb the steps. The slow way home is a *minimum-cost trajectory under a non-time-minimal cost*: she is optimizing for something other than time, and the cost functional rewards the river path even though it is longer.

Same kid. Same bike. Same physics. Four different cost functionals. Four different geodesics on four different cost manifolds, all sitting on the same state space.

This chapter walks the four classical guidance laws that the aerospace literature names for these four problem shapes. Waypoint following (§22.1), proportional navigation for pursuit (§22.2), terminal energy management (§22.3), and minimum-time and minimum-fuel optimization (§22.4). Each is a special case of the cost-manifold geodesic computation from Chapter 22. The framework's contribution is the structural reading that makes the four laws visibly instances of the same machinery—with the cost functional being what distinguishes one from another, and the Davis Duality (Chapter 1 §1.3) bounding the linearization error in each case.

§ 22.1

## *Waypoint following*

The library, then the store, then her friend's house. Three goals. The vehicle visits them in order; the trajectory between any two consecutive goals is the geodesic from the previous to the next, with the previous goal's terminal state as the initial condition for the next segment.

**The classical setup.** Given a sequence of waypoints $\{p_1, p_2, \dots, p_n\}$ in the configuration space, find a trajectory $x(t)$ that passes through each waypoint in order and minimizes a total cost (typically a weighted sum of time, control effort, and smoothness). The waypoints are often required to be visited with specific arrival velocities, attitudes, or other terminal conditions; the segments between waypoints are then geodesics on the cost manifold with the corresponding boundary conditions.

**Practical realities.** Hard waypoint conditions (arrive at exactly $p_i$ at exactly $t_i$) lead to trajectories with discontinuous higher derivatives at the waypoints—the vehicle has to "jerk" at each waypoint to switch from one segment's velocity to the next segment's velocity. The standard fix is to relax the hard conditions: allow the trajectory to pass near the waypoint within a tolerance ball, with the cost penalizing deviation, and require continuity of higher derivatives across the segments. The minimum-snap trajectory generation of Mellinger and Kumar ("Minimum snap trajectory generation and control for quadrotors," *ICRA* 2011) is the canonical reference for the quadrotor case: minimize the integrated fourth derivative of position subject to waypoint and obstacle constraints, with the resulting trajectory expressed in closed form as piecewise polynomial of degree seven.

**The framework reading.** The cost manifold for waypoint following carries a Riemannian-style metric induced by the smoothness penalty (the integrated fourth derivative, in the Mellinger-Kumar formulation). Minimum-snap trajectories are stationary curves of this smoothness functional, and they admit a geodesic reading in an appropriate path-space metric under additional assumptions (the smoothness functional pulls back to a metric on the configuration manifold via the Mellinger-Kumar polynomial parameterization). Under that reading, the waypoints are constraints—specific points the curve must pass through—and the segments between waypoints are unconstrained stationary curves with the waypoint conditions as boundary data. Computing $\hat{\kappa}$ on this manifold tells the planner whether the closed-form polynomial solution is structurally optimal (low curvature, the linearization is right) or whether a nonlinear trajectory optimization is needed (high curvature, e.g., when obstacles or aerodynamic constraints make the cost manifold non-flat).

§ 22.2

## *Proportional Navigation: pursuit*

The dog-chase. The kid is steering toward a target whose future position depends on the target's own decisions, not just on the kid's velocity. The target is non-cooperative: she does not get to schedule the meet-up at a fixed location. She has to converge on the target's actual trajectory, which she sees through the angle between her current heading and the line connecting her to the dog (the *line-of-sight*, or LOS).

**The classical guidance law.** Proportional Navigation (PN) commands lateral acceleration proportional to the rotation rate

of the line-of-sight:

$$a_{\text{cmd}} \;=\; N \cdot V_c \cdot \dot{\sigma},$$

where $\dot{\sigma}$ is the LOS angular rate, $V_c$ is the closing speed, and $N$ is the *navigation constant* (typically chosen in the range $3 \le N \le 5$). The law's intuition: if the LOS angle to the target is rotating, the kid is not on an intercept course; the commanded lateral acceleration corrects the heading until the LOS rate goes to zero, which is the condition for a collision course. PN was developed for missile guidance in the 1940s and 1950s (the canonical reference is Zarchan, *Tactical and Strategic Missile Guidance*, AIAA, 1990 and later editions); it is also the classical law for any pursuit problem where one vehicle must converge on another's trajectory.

**The framework reading.** The cost manifold for pursuit has a metric induced by predicted miss distance at the future intercept. PN is the classical linearized guidance law for this problem: it is optimal under the standard idealized pursuit assumptions (small LOS angles, constant closing speed, non-maneuvering target, unconstrained lateral acceleration), and it is structurally close to the true minimum-miss trajectory in that regime. "Exact" is the wrong word—PN is the cleanest linearization, not the literal geodesic, and the literal geodesic requires the full Pontryagin analysis on the constrained pursuit problem. Outside this regime, PN's solution is structurally bounded away from the actual geodesic by the Davis Duality. In high-curvature regions (the target maneuvering, the closing geometry departing from the linear regime, control authority near saturation), the framework prescribes a nonlinear pursuit law—augmented PN, optimal guidance with the target's acceleration estimate, or model-predictive pursuit—and the substrate's $\hat{\kappa}$ on the pursuit cost manifold is the scalar that tells the planner which regime it is in.

**The defensive context.** The mathematics of PN is the same

in every application. What changes between applications is which cost functional is wired to which actuator. In the framework's deployment, PN-like pursuit laws are wired to collision avoidance, formation-flying with safety constraints, and follow-mode operations (search-and-rescue, cinematography, biological observation): the "target" is something to converge on at a safe relative state, not something to engage. The framework's defensive architectural commitment from Chapter 20 §20.4 is what selects which cost-functional / actuator-wiring combinations are admissible in the schema. The pursuit math is the math; the schema is where the application choice lives.

## § 22.3

## *Terminal energy management*

The bus. Twenty seconds, fixed energy budget, the trajectory has to land her at the door with enough left in her legs to climb the steps. The problem has a specific shape that the aerospace literature has worked on since the Apollo program and developed extensively for the Space Shuttle's entry-and-landing phases: arrive at a specified terminal state, with specified terminal conditions on the energy budget, in a specified amount of time, with the trajectory respecting hard physical limits along the way (the Shuttle could not pull more than a few g's of load factor; the kid cannot pedal faster than her legs allow).

**The classical setup.** The total mechanical energy along the trajectory is $E(t) = \frac{1}{2}V^2 + gh$, the sum of kinetic and potential. The terminal-energy-management problem is to drive $E(t_f)$ to a specified $E_{\text{target}}$ at the specified final time $t_f$, while satisfying the dynamics and the constraints. For the Space Shuttle, this

was the energy-vs-range guidance: at each point in the descent, compute the current energy, compare to the energy needed to fly the remaining range, and command a roll (which adjusts the lift vector to bleed or conserve energy through the resulting bank-angle modulation). Harpold and Graves ("Shuttle Entry Guidance," *Journal of the Astronautical Sciences*, 1979) is the canonical reference.

**The framework reading.** The cost manifold for terminal-energy management has a metric induced by the deviation from the target energy profile, integrated over the remaining trajectory. The geodesics on this manifold are the trajectories that converge on $E_{\text{target}}$ smoothly, respecting the constraints. The framework's reading of the Shuttle's energy-management law is that it is a quasi-linear geodesic computation under the (approximately valid) assumption that energy bleed scales linearly with bank-angle commanded over short intervals. Where this assumption holds, the Shuttle law is structurally optimal. Where it fails—during high-load-factor maneuvers, near maximum-lift-to-drag conditions, in dispersed atmospheric density profiles—the linearized law has the Davis Duality lower-bound error, and the framework prescribes a nonlinear update (the Shuttle's actual flight software has multiple modes for exactly this reason).

§ 22.4

## *Minimum-time and minimum-fuel optimization*

The slow way home along the river is the dual case. The kid has nowhere else to be; she optimizes for something other than time. Aerospace has both ends of this duality. *Minimum-time* trajectories arrive at the goal as fast as possible. *Minimum-*

*fuel* trajectories arrive having used as little control effort as possible. The two are not the same and often pull in opposite directions: the time-optimal trajectory uses maximum control authority continuously; the fuel-optimal trajectory uses control as sparingly as possible.

**Minimum time.** Pontryagin's Minimum Principle, applied to the cost functional $J = \int_{t_0}^{t_f} 1\, dt = t_f - t_0$, gives the classical *bang-bang* result for problems with bounded control authority: the optimal control is at the boundary of the admissible control set almost everywhere, switching between extreme values at specific times determined by the costate dynamics. The brachistochrone (Bernoulli's 1696 challenge, this chapter's epigraph) is the founding minimum-time problem of the calculus of variations: a particle descending under gravity from $A$ to $B$ follows the cycloid, a stationary curve of the time-of-descent functional. Modern minimum-time problems with bounded control inputs—fastest climb to altitude, fastest turn-around in a defensive evasion, fastest emergency landing pattern—return bang-bang structure under the corresponding Pontryagin analysis; the brachistochrone itself, as Bernoulli posed it, has no control bound and is a smooth variational extremum rather than a bang-bang trajectory.

**Minimum fuel.** The cost functional is $J = \int_{t_0}^{t_f} \|u(t)\|_1\, dt$ or $\|u\|_2$ or a similar effort norm. Pontryagin gives a different structure: the optimal control is bang-bang *or zero* on each interval, with singular arcs (where the costate identically vanishes along a subinterval) emerging for specific problems. Bryson and Ho (*Applied Optimal Control*, Blaisdell, 1969) is the standard textbook treatment. Minimum-fuel trajectories arise in spacecraft trajectory design (the Apollo translunar coast; the Voyager Grand Tour's gravity assists; modern station-keeping problems), in fuel-optimal cruise (the most-efficient airline routing problem), and in the cost-of-actuator-wear problems that show up in long-duration

unmanned operations.

**The framework reading.** Both problems are optimal-control problems that admit geometric interpretations as variational problems on the configuration manifold equipped with a cost-induced structure. Minimum-time is naturally read as a *sub-Riemannian* (or sub-Finsler) problem: the admissible velocity distribution is determined by the dynamics, and the minimum time between two points is the distance in the sub-Riemannian metric the dynamics induce. Minimum-fuel with an $L^p$ effort cost is read as a Finsler-style variational problem; the $L^1$ case is the standard fuel cost and produces bang-off-bang structure with singular arcs, while the $L^2$ effort cost produces smooth controls (and reduces to a Riemannian metric in benign settings). These are not clean Riemannian metrics in the strict differential-geometric sense—calling $ds^2 = dt^2$ a metric is looser than the formal sub-Riemannian construction—but the framework's general reading (variational structure on a cost-induced geometry, with the Davis Duality bounding the linearization) specializes cleanly to both cases through the cost functional's choice. The minimum-time bang-bang structure and the minimum-fuel singular arcs are features of the optimal controls on those specific problems, recoverable from the geometric formulation by the Pontryagin analysis.

§ 22.5

## *One machinery, many cost functionals*

The chapter has walked four classical guidance laws. The framework reads them as four different cost functionals on the same state space, producing four different cost manifolds, producing

four different geodesics. The geometric machinery is unchanged across the four cases; the cost functional is what selects which problem the geodesic solves.

**The substrate's reading.** ICARUS's planner (Chapter 20 §20.5) reads the mission specification, which includes the goal, the cost functional, and the constraints. From the cost functional, the planner constructs the cost manifold's metric. From the metric, the planner queries the substrate for $\hat{\kappa}$ along the candidate trajectory. From $\hat{\kappa}$, the planner selects the solver: closed-form polynomial for low-curvature waypoint following, PN for low-curvature pursuit, the Shuttle-style quasi-linear update for low-curvature terminal energy management, Pontryagin bang-bang or singular arcs for the optimization problems. When $\hat{\kappa}$ is high—when the manifold's curvature implies the linearization is structurally inadequate—the planner falls back to a nonlinear method (direct collocation, differential dynamic programming, iterative LQR, or model-predictive control with the cost functional as the objective).

**The defensive commitment, again.** The architectural commitment from Chapter 20 §20.4 is what makes the cost-functional choice an admissible operation in the schema. ICARUS's planner can plan a waypoint-following trajectory through GPS coordinates that the mission system specifies, a pursuit trajectory toward a tracked object that the perception system identifies, a terminal energy management trajectory for the landing phase that the mission system schedules, and a minimum-fuel cruise trajectory that the long-range planner computes. It cannot plan a trajectory whose terminal condition is the engagement of an offensive actuator that the schema does not admit. The cost-functional landscape that the planner has access to is bounded by the framework's defensive architectural commitment, not by the geometric machinery, which is invariant to that choice.

**The mathematical content.** What this chapter has established is that the framework's central operational claim from Chapter 22—that guidance is geodesic computation on a cost manifold, with LQR-style linearization as the flat-tangent approximation and the Davis Duality bounding the linearization error—specializes cleanly to the four classical guidance laws. Each law is the geodesic on a specific cost manifold under a specific linearization. The framework's $\hat{\kappa}$ for each manifold is what tells the planner whether the classical law suffices or whether a nonlinear extension is needed. The four laws are not four separate algorithms; they are four cost-functional instances of one geodesic-computation machinery.

## § 22.6

## *What you take from this chapter*

Four things.

**First.** Waypoint following is the geodesic computation on a cost manifold whose metric is induced by smoothness or effort penalties between specified waypoints. The Mellinger-Kumar minimum-snap formulation (*ICRA* 2011) is the canonical closed-form solution for quadrotors when the linearization is structurally adequate.

**Second.** Proportional Navigation $a_{\text{cmd}} = N \cdot V_c \cdot \dot{\sigma}$ is the classical linearized guidance law for pursuit, optimal under standard idealized engagement assumptions (small LOS angles, constant closing speed, non-maneuvering target, unconstrained lateral acceleration) and structurally bounded away from the true minimum-miss trajectory outside that regime. The Davis Duality bounds the PN-vs-optimal-pursuit error in terms of the

cost manifold's curvature. In ICARUS, PN-class laws are wired to collision avoidance and follow-mode, not to engagement; the architectural schema from Chapter 20 §20.4 is what enforces that choice.

**Third.** Terminal energy management is the geodesic on a cost manifold whose metric is induced by deviation from a target energy profile. The Space Shuttle's entry-guidance algorithm (Harpold-Graves 1979) is the classical instance; the framework reads it as a quasi-linear geodesic computation with the Davis Duality lower-bound error in high-load-factor and near-max-lift-to-drag regimes.

**Fourth.** Minimum-time and minimum-fuel optimization are geodesics on cost manifolds with the specific metrics $ds^2 = dt^2$ (sub-Riemannian) and $ds^2 = \|u\|^p\, dt$ (Finsler). The Pontryagin bang-bang structure for minimum-time and the singular-arc structure for minimum-fuel are recoverable from the geometric formulation by the standard Pontryagin analysis. The brachistochrone is the founding instance; modern problems return the same structural answers under the same Pontryagin machinery.

The next chapter is the methodological close: the *Sudoku Principle*, the framework's epistemic discipline named formally and walked carefully. The book has been operating under it for twenty-two chapters; the chapter coming next is where the operation gets a name.

*Vamos.*

# *Lab VI: Guidance on the cost manifold — linearized vs nonlinear, with honest scaling.*

*Time: about 60–75 minutes. Materials: Python 3.10 or later with* `numpy`*,* `scipy`*,* `matplotlib`*, and* `dhoom`*.*

Lab V exercised the controller-and-substrate inner loop. This lab runs the outer guidance loop: solve a 2D trajectory-generation problem three different ways, then check whether the framework's structural prediction (linearized solvers' error against the nonlinear solver scales with the cost-manifold's curvature) holds across two obstacle configurations.

**What this lab demonstrates, honestly.** The lab compares three solvers on a 2D point-mass guidance problem. It does *not* compute a numerical Davis Duality lower bound—deriving the actual constant for this specific cost functional is open work, and a magic-number bound would be unfalsifiable. Instead, the lab performs a *structural scaling check*: empirical error of the linearized solvers (A and B) against the nonlinear solver (C) should be small when $\widehat{\kappa}$ on the cost manifold is small, and large when $\widehat{\kappa}$ is large. The framework's reading from Chapter 22 §21.3 predicts the direction of the scaling; the lab verifies that the

direction is right.

**The problem we'll use.** A 2D point-mass at rest at the origin must reach $(10, 0)$ in 5 seconds with zero terminal velocity. There is a single circular obstacle; we run the lab twice, once with the obstacle off the direct path (low cost-manifold curvature) and once with the obstacle on the direct path (high cost-manifold curvature).

Step 1

## *Set up*

```
mkdir lab6 && cd lab6
python  # or jupyter notebook

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
import dhoom
```

Step 2

## *The problem statement and the straight-line reference*

```
# 2D point mass. State (x, y), control (ax, ay).
T = 5.0
N = 50
dt = T / N
```

```
start_pos = np.array([0.0, 0.0])
goal_pos  = np.array([10.0, 0.0])
delta = 0.5   # safety margin around the obstacle

# Two scenarios: easy (obstacle off-path) and hard (obstacle on-path).
obstacle_easy = {'center': np.array([5.0, 4.0]), 'radius': 1.5}
obstacle_hard = {'center': np.array([5.0, 0.0]), 'radius': 1.5}

# The straight-line reference is the geodesic on the FLAT cost manifold
# (no obstacle). All three solvers are compared against it for the
# scaling analysis; Solvers B's linearization is taken around it.
def straight_line_reference(start, goal, N):
    return np.linspace(start, goal, N+1)
```

Step 3

## *Solver A: closed-form minimum-snap (no obstacle awareness)*

```
def solve_min_snap(start, goal, T, N):
    # Closed-form min-snap polynomial (Mellinger-Kumar 2011).
    # Boundary conditions: position = start/goal, derivatives 1..3 = 0
    # at both endpoints. Result: degree-7 polynomial in each axis,
    # uniquely determined by the 8 conditions.
    import math
    def poly_coeffs(p0, pf):
        A = np.zeros((8, 8))
        b = np.zeros(8)
        for k in range(4):
            A[k, k] = math.factorial(k)
            b[k] = p0 if k == 0 else 0
            for i in range(k, 8):
                A[4+k, i] = math.factorial(i) // math.factorial(i-k)
            b[4+k] = pf if k == 0 else 0
        return np.linalg.solve(A, b)
    cx = poly_coeffs(start[0], goal[0])
    cy = poly_coeffs(start[1], goal[1])
    tau = np.linspace(0, 1, N+1)
    xs = sum(cx[i] * tau**i for i in range(8))
```

```
    ys = sum(cy[i] * tau**i for i in range(8))
    return np.column_stack([xs, ys])

traj_A_easy = solve_min_snap(start_pos, goal_pos, T, N)
traj_A_hard = traj_A_easy.copy()   # solver ignores obstacles
print('Solver A (min-snap) trajectories computed.')
```

Step 4

## *Solver B: linearized QP with dynamics-coupled smoothness*

```
def solve_linearized(start, goal, T, N, obstacle, delta):
    # Single-shot QP that linearizes the soft barrier around the
    # straight-line reference. The linearization is genuine:
    #   1. Smoothness: integrated squared finite-difference acceleration
    #      (couples consecutive timesteps -- not a per-point penalty).
    #   2. Barrier: linearized about the straight-line reference, so the
    #      quadratic approximation is exact at the reference and grows
    #      quadratically with the deviation along the obstacle normal.
    #   3. Boundary conditions: equality constraints at start, goal,
    #      and zero-velocity endpoints.
    # The QP is solved as a KKT linear system. No iteration --
    # this is the "one linearization step" version of the framework's
    # flat-tangent approximation from Chapter 21 §21.3.
    n = (N+1) * 2
    ref = straight_line_reference(start, goal, N)
    barrier_radius = obstacle['radius'] + delta

    # Smoothness Hessian: integrated squared acceleration.
    # accel_k = (p_{k+1} - 2 p_k + p_{k-1}) / dt^2, k = 1..N-1.
    # Cost = sum_k ||accel_k||^2 * dt.
    H_smooth = np.zeros((n, n))
    # The QP is min (1/2) z^T H z + g^T z, so H must be the full Hessian
    # of the cost. The cost is J = sum_k ||accel_k||^2 * dt, with
    # accel_k = (p_{k+1} - 2 p_k + p_{k-1}) / dt^2. The full Hessian of
    # (a)^2 with respect to its variables is 2 * (gradient outer product),
    # hence the factor of 2.
    coef = 2.0 * dt / (dt**4)   # factor 2 from (1/2) z^T H z form
```

```
for k in range(1, N):
    for ax in range(2):
        i_prev, i_curr, i_next = 2*(k-1)+ax, 2*k+ax, 2*(k+1)+ax
        H_smooth[i_prev, i_prev] += coef
        H_smooth[i_curr, i_curr] += 4 * coef
        H_smooth[i_next, i_next] += coef
        H_smooth[i_prev, i_curr] += -2 * coef
        H_smooth[i_curr, i_prev] += -2 * coef
        H_smooth[i_prev, i_next] += coef
        H_smooth[i_next, i_prev] += coef
        H_smooth[i_curr, i_next] += -2 * coef
        H_smooth[i_next, i_curr] += -2 * coef

# Linearized barrier:
#   phi(p) = K_b * max(0, barrier_radius - normal^T(p - center))^2
# Linearized around ref_k where ref_k is in the barrier zone.
H_barrier = np.zeros((n, n))
g_barrier = np.zeros(n)
K_b = 50.0
motion = goal - start
motion_perp = np.array([-motion[1], motion[0]])
motion_perp = motion_perp / np.linalg.norm(motion_perp)
for k in range(N+1):
    diff = ref[k] - obstacle['center']
    d = np.linalg.norm(diff)
    if d < barrier_radius:
        # Choose outward normal at the reference point.
        normal = diff / d if d > 1e-3 else motion_perp
        # Linearization (around ref):
        #   phi(p) ~ K_b * (barrier_radius - normal^T p + n^T c)^2
        # Quadratic in p:
        #   Hessian = 2 K_b normal normal^T
        #   linear  = -2 K_b (barrier_radius + n^T center) * normal
        idx_x, idx_y = 2*k, 2*k+1
        nn = np.outer(normal, normal)
        H_barrier[idx_x:idx_x+2, idx_x:idx_x+2] += 2 * K_b * nn
        lin = -2 * K_b * (barrier_radius
               + normal @ obstacle['center']) * normal
        g_barrier[idx_x] += lin[0]
        g_barrier[idx_y] += lin[1]

H = H_smooth + H_barrier + np.eye(n) * 1e-8   # tiny regularizer

# Boundary conditions: p_0 = start, p_N = goal,
# p_1 - p_0 = 0 (zero initial vel),
# p_N - p_{N-1} = 0 (zero terminal vel).
n_eq = 8
```

```
    A_eq = np.zeros((n_eq, n))
    b_eq = np.zeros(n_eq)
    A_eq[0, 0] = 1.0; b_eq[0] = start[0]
    A_eq[1, 1] = 1.0; b_eq[1] = start[1]
    A_eq[2, 2*N] = 1.0; b_eq[2] = goal[0]
    A_eq[3, 2*N+1] = 1.0; b_eq[3] = goal[1]
    A_eq[4, 0] = -1.0; A_eq[4, 2] = 1.0
    A_eq[5, 1] = -1.0; A_eq[5, 3] = 1.0
    A_eq[6, 2*N] = 1.0;     A_eq[6, 2*(N-1)] = -1.0
    A_eq[7, 2*N+1] = 1.0;   A_eq[7, 2*(N-1)+1] = -1.0

    # KKT solve: [[H, A^T], [A, 0]] [z; lam] = [-g; b].
    KKT = np.block([[H, A_eq.T],
                    [A_eq, np.zeros((n_eq, n_eq))]])
    rhs = np.concatenate([-g_barrier, b_eq])
    sol = np.linalg.solve(KKT, rhs)
    return sol[:n].reshape(N+1, 2)

traj_B_easy = solve_linearized(
    start_pos, goal_pos, T, N, obstacle_easy, delta)
traj_B_hard = solve_linearized(
    start_pos, goal_pos, T, N, obstacle_hard, delta)
print('Solver B (linearized QP) trajectories computed.')
```

Step 5

## *Solver C: nonlinear collocation with terminal-velocity BC*

```
def solve_collocation(start, goal, T, N, obstacle):
    # Direct collocation: discretize, solve as constrained NLP using
    # SLSQP. The obstacle is enforced as a hard nonlinear inequality
    # constraint d(p_k) >= radius. Boundary conditions match Solver A:
    # start and goal positions plus zero initial and terminal velocity
    # (enforced as p_1 = p_0 and p_N = p_{N-1} in finite-difference form).
    z0 = np.linspace(start, goal, N+1).flatten()

    def cost(z):
        traj = z.reshape(N+1, 2)
```

```
        a = (traj[2:] - 2*traj[1:-1] + traj[:-2]) / dt**2
        return float(np.sum(a**2)) * dt

    def obstacle_constraint(z):
        traj = z.reshape(N+1, 2)
        d = np.linalg.norm(traj - obstacle['center'], axis=1)
        return d - obstacle['radius']            # >= 0

    def boundary_constraint(z):
        traj = z.reshape(N+1, 2)
        # Positions:
        bc_pos = np.concatenate([traj[0] - start, traj[-1] - goal])
        # Zero initial and terminal velocity (finite-difference form):
        bc_vel = np.concatenate([traj[1] - traj[0], traj[-1] - traj[-2]])
        return np.concatenate([bc_pos, bc_vel])

    constraints = [
        {'type': 'ineq', 'fun': obstacle_constraint},
        {'type': 'eq',   'fun': boundary_constraint},
    ]
    result = minimize(cost, z0, method='SLSQP',
                      constraints=constraints,
                      options={'maxiter': 200, 'ftol': 1e-5})
    return result.x.reshape(N+1, 2)

traj_C_easy = solve_collocation(start_pos, goal_pos, T, N, obstacle_easy)
traj_C_hard = solve_collocation(start_pos, goal_pos, T, N, obstacle_hard)
print('Solver C (nonlinear collocation) trajectories computed.')
```

Step 6

## *Design-choice $\widehat{\kappa}$: a property of the cost manifold, not of any trajectory*

```
def kappa_on_manifold(start, goal, N, obstacle, delta):
    # Design-choice kappa_hat for the cost manifold. This is a property
    # of the *problem instance* (start/goal/obstacle), evaluated on the
    # straight-line reference (the flat-ambient geodesic). It is the
    # same value for all three solvers in the same scenario, because
```

```
    # it characterizes the manifold's local curvature near the reference
    # path, not any particular solver's output.
    #
    # The form: integrated squared barrier-overlap along the reference,
    # normalized by the path length. This is monotone non-negative and
    # zero when the reference does not enter the barrier zone (Chapter
    # 10 §10.4 design-choice family; motivated by Chapter 21 §21.5's
    # reading of obstacles as high-curvature regions of the manifold).
    ref = straight_line_reference(start, goal, N)
    barrier_radius = obstacle['radius'] + delta
    overlap_sq = 0.0
    for k in range(N+1):
        d = np.linalg.norm(ref[k] - obstacle['center'])
        overlap_sq += max(0.0, barrier_radius - d) ** 2
    path_length = np.linalg.norm(goal - start)
    return overlap_sq / max(path_length, 1e-6)

kappa_easy = kappa_on_manifold(
    start_pos, goal_pos, N, obstacle_easy, delta)
kappa_hard = kappa_on_manifold(
    start_pos, goal_pos, N, obstacle_hard, delta)
print(f'kappa_hat (easy, obstacle off-path) = {kappa_easy:.4f}')
print(f'kappa_hat (hard, obstacle on-path)  = {kappa_hard:.4f}')
```

Step 7

## *Structural scaling check (replaces the magic-number lower bound)*

```
# The framework's Davis Duality reading (Chapter 1 §1.3, Chapter 18 §18.4)
# says: the error of a flat-tangent approximation against the true geodesic
# is BOUNDED BELOW by a multiple of (integrated curvature) * (step size)^2.
# Deriving the multiplicative constant for this specific cost functional is
# open work. What the lab CAN check, falsifiably,
# is the structural scaling:
# the linearized solvers' error against the nonlinear solver should be
# small when kappa_hat is small and large when kappa_hat is large.
#
# Concretely: compute the mean position error of each linearized solver
```

```
# against Solver C, and report the absolute contrast (error_hard -
# error_easy) and a Pearson correlation across multiple obstacle
# offsets.  Reporting a hard/easy ratio is misleading because the
# easy scenario is designed so kappa_easy is essentially zero; a
# ratio against a near-zero denominator is arbitrary.  The framework's
# directional prediction is: larger kappa-hat corresponds to larger
# linearization error.  We test that with a sweep of obstacle
# offsets, not a single ratio.
def mean_error(traj_approx, traj_true):
    return float(np.linalg.norm(traj_approx - traj_true, axis=1).mean())

err_A_easy = mean_error(traj_A_easy, traj_C_easy)
err_A_hard = mean_error(traj_A_hard, traj_C_hard)
err_B_easy = mean_error(traj_B_easy, traj_C_easy)
err_B_hard = mean_error(traj_B_hard, traj_C_hard)

print()
print('Structural scaling check (two scenarios):')
print(f'  kappa contrast (hard - easy)'
      f' = {kappa_hard - kappa_easy:+.4f}')
print(f'  Solver A error contrast (hard - easy)'
      f' = {err_A_hard - err_A_easy:+.4f}')
print(f'  Solver B error contrast (hard - easy)'
      f' = {err_B_hard - err_B_easy:+.4f}')
print()
print(f'  Directional check: both contrasts should be positive')
print(f'  (larger kappa => larger linearization error).')
print()
# Optional extension: sweep obstacle offsets in y to get many
# (kappa, error) pairs, then compute the Pearson correlation.
# That converts the directional check into a quantitative monotonic
# trend test rather than a fragile two-point ratio.
offsets = np.linspace(0.0, 4.0, 9)
kappas, errs_A, errs_B = [], [], []
for dy in offsets:
    obs = {'center': np.array([5.0, dy]), 'radius': 1.5}
    k = kappa_on_manifold(start_pos, goal_pos, N, obs, delta)
    tA = traj_A_easy.copy()
    tB = solve_linearized(start_pos, goal_pos, T, N, obs, delta)
    tC = solve_collocation(start_pos, goal_pos, T, N, obs)
    kappas.append(k)
    errs_A.append(mean_error(tA, tC))
    errs_B.append(mean_error(tB, tC))
kappas = np.array(kappas)
errs_A = np.array(errs_A)
errs_B = np.array(errs_B)
rA = float(np.corrcoef(kappas, errs_A)[0, 1])
```

```
rB = float(np.corrcoef(kappas, errs_B)[0, 1])
print(f'Sweep correlation (across {len(offsets)} offsets):')
print(f'  Pearson r(kappa, err_A) = {rA:+.3f}')
print(f'  Pearson r(kappa, err_B) = {rB:+.3f}')
print()
print(f'  Empirical errors at the two named scenarios:')
print(f'    Solver A vs C: easy={err_A_easy:.4f},'
      f' hard={err_A_hard:.4f}')
print(f'    Solver B vs C: easy={err_B_easy:.4f},'
      f' hard={err_B_hard:.4f}')
```

Step 8

*Plot the three solvers against the two scenarios*

```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

def draw_circle(ax, center, radius, color='red', alpha=0.3, label=None):
    theta = np.linspace(0, 2*np.pi, 100)
    ax.fill(center[0] + radius*np.cos(theta),
            center[1] + radius*np.sin(theta),
            color=color, alpha=alpha, label=label)

# Easy-scenario trajectories.
ax = axes[0]
ax.plot(traj_A_easy[:, 0], traj_A_easy[:, 1], 'b-',
        label='A: min-snap (ignores obs)')
ax.plot(traj_B_easy[:, 0], traj_B_easy[:, 1], 'g--',
        label='B: linearized QP')
ax.plot(traj_C_easy[:, 0], traj_C_easy[:, 1], 'r-',
        label='C: nonlinear NLP')
draw_circle(ax, obstacle_easy['center'], obstacle_easy['radius'],
            color='red',    alpha=0.3, label='obstacle')
draw_circle(ax, obstacle_easy['center'], obstacle_easy['radius']+delta,
            color='orange', alpha=0.15, label='barrier zone')
ax.scatter(*start_pos, c='black', marker='o', s=80, label='start')
ax.scatter(*goal_pos,  c='black', marker='*', s=120, label='goal')
ax.set_aspect('equal'); ax.legend(loc='upper right', fontsize=8)
ax.set_title(rf'Easy (off-path): $\widehat{{\kappa}} = {kappa_easy:.3f}$')
ax.grid(True)
```

```
# Hard-scenario trajectories.
ax = axes[1]
ax.plot(traj_A_hard[:, 0], traj_A_hard[:, 1], 'b-',
        label='A: min-snap (ignores obs)')
ax.plot(traj_B_hard[:, 0], traj_B_hard[:, 1], 'g--',
        label='B: linearized QP')
ax.plot(traj_C_hard[:, 0], traj_C_hard[:, 1], 'r-',
        label='C: nonlinear NLP')
draw_circle(ax, obstacle_hard['center'], obstacle_hard['radius'],
            color='red',    alpha=0.3, label='obstacle')
draw_circle(ax, obstacle_hard['center'], obstacle_hard['radius']+delta,
            color='orange', alpha=0.15, label='barrier zone')
ax.scatter(*start_pos, c='black', marker='o', s=80, label='start')
ax.scatter(*goal_pos,  c='black', marker='*', s=120, label='goal')
ax.set_aspect('equal'); ax.legend(loc='upper right', fontsize=8)
ax.set_title(rf'Hard (on-path): $\widehat{{\kappa}} = {kappa_hard:.3f}$')
ax.grid(True)

plt.tight_layout(); plt.savefig('lab6_results.png', dpi=120)
plt.show()
```

Step 9

## *What you should see, and what it actually demonstrates*

**Easy scenario (obstacle off-path).** $\widehat{\kappa}$ is approximately zero because the straight-line reference does not enter the obstacle's barrier zone. All three solvers produce nearly identical trajectories: a smooth path roughly along the straight line. Solver A's error against Solver C is small. Solver B's error against Solver C is small. This is the framework's low-curvature regime: the linearization is structurally adequate, and the cheap solvers are doing the same work as the expensive one.

**Hard scenario (obstacle on-path).** $\widehat{\kappa}$ is non-zero (substantially larger than the easy case) because the straight-line refer-

ence passes through the obstacle's barrier zone. Solver A (which ignores obstacles) drives straight through the obstacle—its trajectory is the same as in the easy scenario. Solver B (the linearized QP) deflects perpendicular to the straight-line reference; the deflection's magnitude depends on the linearization's barrier strength, and depending on $K_b$ may clear the obstacle, may skim it, or may over-deflect. Solver C (the nonlinear NLP) routes around the obstacle exactly, hugging the safety boundary. The errors of Solvers A and B against C are both substantially larger than in the easy scenario.

**The structural scaling check.** The ratio (kappa_hard / kappa_easy) measures how much more curved the cost manifold is in the hard configuration. The ratios (err_A_hard / err_A_easy) and (err_B_hard / err_B_easy) measure how much more error the linearized solvers make in the hard configuration. The framework's reading from Chapter 22 §21.3 predicts that the error ratios should be in the same direction as the $\widehat{\kappa}$ ratio: more curvature means more linearization error. If the printed scaling lines show this direction (error ratios all $> 1$ when the kappa ratio is $> 1$), the framework's structural prediction is consistent with the lab's evidence. If not, the framework's prediction is falsified for this configuration.

**What the lab does NOT do.** It does *not* compute a numerical Davis Duality lower bound. The framework's lower-bound theorem (Davis 2026, Zenodo DOI `10.5281/zenodo.19428406`) involves a multiplicative constant that depends on the specific cost functional, the manifold's local geometry, and the trajectory's parameterization. Deriving that constant for this lab's soft-barrier formulation is open work. A magic-number bound would be unfalsifiable—tunable to whatever the empirical numbers happen to be—so the lab declines to print one. The structural scaling check is what is falsifiable, and that is what the lab

prints.

**The substrate reading.** ICARUS's planner reads the cost manifold's $\widehat{\kappa}$ for the candidate problem instance *before committing to a solver*. When the easy-scenario $\widehat{\kappa}$ comes back small, the planner runs the cheap Solver A or B. When the hard-scenario $\widehat{\kappa}$ comes back large, the planner falls back to Solver C. The framework's substrate-level confidence scalar tells the planner which solver to use; the lab demonstrates that the scalar predicts the right answer on these two instances.

Step 10

## *Store the run as DHOOM*

```
# DHOOM bundle: time-base section list with scenario and solver
# discriminators, matching the list-of-records pattern from Labs IV and V.
# DHOOM bundles require exactly one top-level key (the bundle name).
# Records are a flat list with scenario/solver/k discriminator fields.
guidance_records = []
for scenario_name, traj_set in [
    ('easy', [('A', traj_A_easy), ('B', traj_B_easy), ('C', traj_C_easy)]),
    ('hard', [('A', traj_A_hard), ('B', traj_B_hard), ('C', traj_C_hard)]),
]:
    for solver_name, traj in traj_set:
        for k in range(len(traj)):
            guidance_records.append({
                'scenario': scenario_name,
                'solver': solver_name,
                'k': int(k),
                't': float(k * dt),
                'x': float(traj[k, 0]),
                'y': float(traj[k, 1]),
            })
guidance_bundle = {'guidance_run': guidance_records}
with open('lab6_guidance_run.dhoom', 'w') as f:
    f.write(dhoom.encode(guidance_bundle))
print('Guidance run serialized to lab6_guidance_run.dhoom')
```

The serialized bundle is a single trajectory-index section list with `scenario` and `solver` discriminator fields. The header carries the two $\widehat{\kappa}$ values so a future GIGI consumer can compute the same scaling check from the bundle alone.

This closes Part VII. The next chapter is the methodological close: the *Sudoku Principle*, the framework's epistemic discipline named formally and walked carefully. The book has been operating under it for twenty-two chapters; the chapter coming next is where the operation gets a name.

Chapter 24

# *The Sudoku Principle*

*Mathematics is the art of giving the same name*
*to different things.*
*Henri Poincaré, Science et Méthode (1908),*
*Book II, Chapter II*

§ 23.0

## *The kid with a puzzle*

The kid is on the kitchen floor with a newspaper folded open to the back page. The back page has a Sudoku puzzle. Nine squares by nine. Most of the cells are empty; a few have numbers already filled in. She has a pencil. She is not guessing.

She looks at the top-left $3 \times 3$ block. It already has a 5, a 2, an 8. The numbers 1, 3, 4, 6, 7, 9 are missing from this block. She looks down the leftmost column. The leftmost column has a 1, a 5, a 7, a 9. She looks across the top row. The top row has a 5, a 2. From the intersection she works out which cell in the top-left block can hold which missing digit. One cell is forced: the rules permit only 6. She writes 6 in pencil.

Then she does it again. With the 6 written down, more cells

become forced. She writes them in. Each move tightens the constraints. Each constraint, propagated, determines more cells. The puzzle is not solved by guessing; it is solved by writing down what is already determined, and asking, at each step, *what else does this determine?*

This chapter is about that move. The book has been doing it for twenty-three chapters; this chapter names it formally. The full **Sudoku Principle**—with its catalog of convergence types that, together, constitute what *proof* means in the Davis-Geometric framework—is stated formally in §23.7. The sections before that build up to the statement: the seed form of the move (§23.1), the catalog of where the book has been using it (§23.2), the four convergence types (§23.3), the structural-versus-literal distinction the framework holds throughout (§23.4), the failure modes (§23.5), and a worked example—the Davis Duality of Approximation and Obstruction read in full (§23.6). The principle (§23.7) and what you take (§23.8) close the chapter and the methodological arc.

§ 23.1

## *The seed move*

The seed Sudoku move is one question: *if this is true, what else follows?*

Stated more precisely: given a set of established facts and a structural relation among them, ask what additional facts the structure determines. The question is mechanical. The answer is a propagation: the structure is doing the work, the solver is reading the structure honestly.

**What the seed move is not.** The seed move is not a guess.

It is not an assumption. It is not a heuristic. The move only writes down what the structure already entails. If the structure does not entail a fact, the move does not write it down. The discipline is in the refusal to add what is not there.

**Where the move shows up in derivation.** Chapter 1 §1.2 used the seed move to derive the lawnmower-engine-bike's recovery requirement: *if I am going to ride this, what authority do I need to stop?* The question's answer was determined by the structure (the bike's mass, the engine's power, the brake pads' sizing); the kid wrote it down honestly and chose not to ride. Chapter 6 used the seed move to read the F-16's gauge group: *if these state variables admit this coordinate change, what other variables admit the same change?* The answer was determined by the linearization's invariance properties. Chapter 15 used it to extend the navigation bundle: *if INS is a bundle and GNSS is a bundle, what is the structure that contains both?* The coproduct construction was the answer.

In each case the move is the same. Read the structure. Ask what follows. Write it down. Iterate.

§ 23.2

## *The book has been doing this*

The book's twenty-three chapters can be read, in retrospect, as a catalog of the Sudoku move applied to twenty-two specific structural questions. A non-exhaustive list:

**Chapter 1.** If a chain ring on a bicycle and a blade on a lawnmower are both circles in motion, what does that structural identity entail? The Bra Strap Principle: a single local

check determines the section globally on a base space that is appropriately connected.

**Chapter 2.** If the Yang–Mills spectrum has a gap and the gap has the form of a curvature tax, what does the equation $C = \tau/K = 1/(1+\widehat{\kappa})$ entail when read across other manifolds? The Davis Field Equation's reading of the Navier–Stokes regularity obstruction, the P-versus-NP separation, the Poincaré topology, the Riemann zeros.

**Chapter 8.** If the Bellman value function's Hessian defines a Riemannian metric on the state manifold, what does the LQR closed loop look like in that metric? Gradient descent on the value function—structurally, the same operation Pontryagin's principle prescribes in canonical form.

**Chapter 14.** If the IMU's measurement is a connection 1-form on the inertial bundle, what does parallel transport along the time axis compute? The strapdown algorithm.

**Chapter 16.** If loop closure is a single local re-observation propagating a global correction, what is the structural object the substrate must maintain to support it? The cross-correlation block of the joint pose-and-map covariance.

**Chapter 18.** If serialization can be fiber-bundle-aware (DHOOM), what is the storage layer that does the same? GIGI, with `COMPLETE`, `PROPAGATE`, `CONSISTENCY` as primary operations.

**Chapter 20.** If the controller-substrate interface is a single scalar $\widehat{\kappa}$, what does that interface buy the architecture? Graceful degradation under sensor failure, with the type system enforcing the defensive commitment.

Each chapter is one Sudoku move on a specific structural question, with the structure providing the answer and the chapter walking the propagation honestly. The framework's universality claim

is that the same move, applied across these domains, produces consistent answers because the structure they share is the same structure.

§ 23.3

## *Four convergence types*

The framework's notion of *proof* is not a single thing. It is a catalog of distinct convergence types that, together, constitute the discipline's standard of justification. The framework names four.

**Type 1: Local-to-global propagation.** A single local check, propagated through the structure's connection, determines the section globally. The Bra Strap Principle (Davis 2026, Zenodo DOI `10.5281/zenodo.20273573`) is the formal statement; loop closure in SLAM and the `PROPAGATE` verb in GIGI are the operational instances. When the propagation is licit—when the structure satisfies the principle's preconditions—the local check is sufficient.

**Type 2: Cross-domain reading.** The same equation, read in multiple distinct domains, produces structurally consistent answers in each. The Davis Field Equation $C = \tau/K = 1/(1+\widehat{\kappa})$ read across approximation theory (the Davis Duality), quantum field theory (the Yang–Mills mass gap), control theory (the curvature-gated modulation), and the others Chapter 2 §2.5 lists. Consistency across domains is not a proof of any single instance; it is a structural constraint on the equation's correctness.

**Type 3: Limit convergence to classical.** In the limit where the framework's design-choice $\widehat{\kappa}$ goes to zero, the framework's

predictions reduce to the classical results. LQR is the flat-tangent special case of the cost-manifold geodesic. The Bode integral identity is the limiting case of the band-limited measurement. The Kalman filter is the linear-Gaussian special case of the substrate's estimator. When the limit holds, the framework is consistent with what the classical theory already knows.

**Type 4: Multi-method agreement.** The same problem solved by independent methods agrees. Pontryagin's principle, Bellman's dynamic programming, and the Hamilton-Jacobi-Bellman PDE produce the same optimal trajectory; direct collocation, differential dynamic programming, and iterative LQR converge to the same solution when the cost manifold's curvature is low. Multi-method agreement is empirical evidence that the structure being solved is well-defined; disagreement is diagnostic of a structural issue.

Together these four constitute the framework's standard for what a claim has to satisfy to be carried forward. A single type alone is not sufficient. Local-to-global without cross-domain reading is a special case. Cross-domain without limit convergence is a coincidence. Limit convergence without multi-method agreement is an artifact of the chosen method. The four are independent dimensions of the same structural reality, and the framework's commitment is to verify a claim across the dimensions it touches.

§ 23.4

## *Structural reading versus literal identification*

The book has been careful, at several points, to distinguish a *structural reading* from a *literal mathematical identification.* The distinction is part of the Sudoku discipline.

A structural reading says: *the framework's object plays the role that the classical object plays.* The two objects share a pattern of behavior, satisfy analogous equations under analogous transformations, and produce consistent predictions in the regimes where both are defined. Examples: the cross-correlation block plays the role that holonomy plays (Chapter 16 §16.2); the Pontryagin costate plays the role that the cotangent transport object plays (Chapter 22 §21.4); the Bode integral plays the role that the Gauss–Bonnet integrated curvature plays (Chapter 7 §7.4).

A literal identification says: *the framework's object is the classical object.* The two are the same mathematical entity under the same definitions, with the identification provable as a theorem with a specific proof in the literature. Examples: the minimum-time optimal-control problem with bounded control is sub-Riemannian length minimization (Chapter 23 §22.4); the discrete sheaf Laplacian in GIGI's `COMPLETE` is the sheaf Laplacian of Hansen-Ghrist (Chapter 18 §18.3); the Bryson-Ho minimum-fuel singular arcs are recoverable from Pontryagin's principle by standard analysis (Chapter 23 §22.4).

The framework's commitment, applied through the Sudoku discipline, is to consistently mark which kind of claim is being made. Where a literal identification is available, the framework states it. Where only a structural reading is available, the framework states that, and marks the precise mathematical identification as open work. The book's softening of the holonomy and PE-as-holonomy framings across Chapters 7, 12, 16, and 21 is the application of this discipline.

§ 23.5

## *Failure modes of the Sudoku move*

The Sudoku move can fail in three specific ways. Naming the failure modes is part of the discipline because the move is mechanical, not magical, and the failures are predictable.

**Overreach.** The move claims more than the structure entails. The local check propagates further than its preconditions allow, the cross-domain reading is asserted where the domains do not share the relevant structure, the limit is taken in a regime where the limit does not exist. Overreach is the most common failure of structural reading: the framework's universality claim is stretched past the configurations where it actually holds.

**Undermarking.** The move makes a literal identification that should have been a structural reading, or claims a theorem where it had a design choice. The book's earlier framings of the curvature-gated modulation as a "theorem" rather than a "result" (corrected in Chapter 17 §17.3) were an instance of undermarking. The framework's later corrections were the discipline catching itself.

**Missing the constraint.** The move reads the structure but misses one of the constraints the structure imposes. The puzzle is solved with one missing rule, the solution looks consistent locally, but it fails at the boundary where the missing constraint would have applied. Missing-the-constraint failures often look like overreach but are diagnostically distinct: the issue is incompleteness of the structural reading, not over-extension of a complete one.

The discipline against these failures is the same in each case: state what the structure entails; state what it does not entail; identify which type of convergence the claim relies on; flag open

work where a structural reading awaits its literal identification. The framework's epistemic stance is that claims are carried by the structure that supports them, not by the strength of the language used to state them.

§ 23.6

## *Worked example: the Davis Duality, read carefully*

A single worked example to show the Sudoku move on a load-bearing claim of the framework.

**The claim.** The Davis Duality of Approximation and Obstruction (Davis 2026, Zenodo DOI `10.5281/zenodo.19428406`) says that the error of a flat-tangent approximation against a curved section, on a manifold with non-zero curvature, is bounded above *and* below by a multiple of the integrated curvature times the step size squared. The lower bound is the news: there is a structural error floor that no amount of clever fitting can get below.

**Reading the claim with the Sudoku move.** What does the structure entail?

1. *Type 1 (local-to-global propagation).* The Bra Strap Principle on a connected base space says that a single local statement of the bound propagates globally provided the connection has the right structural property. The Davis Duality's local statement is the inequality on a single chart; the propagation is to the global trajectory integral. Type 1 supports the claim where the base space is connected and the connection is well-behaved.
2. *Type 2 (cross-domain reading).* The same bound appears in approximation theory (the original setting), in quan-

tum field theory (the mass gap), in control theory (the LQR-vs-nonlinear gap in cost-manifold geodesics), and in machine learning (the structural error floor of flat-tangent regressions on curved data manifolds). Cross-domain consistency supports the bound's structural form, even where the specific constant is not known across domains.

3. *Type 3 (limit convergence to classical).* In the limit of zero curvature, the bound reduces to the standard Taylor approximation upper bound. The lower bound trivially goes to zero. Type 3 supports the claim's consistency with the classical theory in the flat limit.

4. *Type 4 (multi-method agreement).* The bound has three independent proof routes in the Davis 2026 paper: an associated-bundle functor argument, a Nash embedding into a higher-dimensional flat space, and an obstruction-theoretic computation using Atiyah characteristic classes. Three routes converging on the same bound is multi-method agreement at the proof level.

**What the structure does not entail.** The Davis Duality's specific multiplicative constant for a given cost functional—which is what would let you compute a numerical lower bound on the approximation error in a specific application—is not stated in the published paper for the general nonlinear case. The constant is open work. Lab VI in this book declines to print a numerical lower bound for exactly this reason: a magic-number constant on the framework's central conjecture would be unfalsifiable.

**What the Sudoku discipline produces.** The Davis Duality is claimed as a theorem (it has three proof routes); it is read across four domains (cross-domain reading); it reduces to the classical Taylor bound in the flat limit (limit convergence); and the missing piece (the multiplicative constant for general cost

functionals) is marked as open. That is the move: state what is supported, identify which convergence types support it, flag what is not yet established. The discipline does not promise more than the structure entails.

§ 23.7

## *The Sudoku Principle, formally*

With the seed move, the catalog of uses, the four convergence types, the structural-versus-literal distinction, the failure modes, and the worked example all in place, the principle's formal statement is now stable.

**The Sudoku Principle.** *In the Davis-Geometric framework, a claim is established when its structural content is supported by one or more of the four convergence types (local-to-global propagation; cross-domain reading; limit convergence to classical; multi-method agreement), with the type of support marked explicitly and the boundary of the claim (where the support fails) named honestly. A claim is overreached when its support is asserted past the regime where the supporting convergence types hold. A claim is undermarked when its support is described as a theorem where it is a structural reading, or as a design choice where it is a theorem. The discipline of the principle is the discipline of marking each claim accurately against its actual structural support.*

**What this commits the framework to.** Every claim in this book—and every claim in the Davis-Geometric series the book is part of—is supported by some combination of the four convergence types, with the support explicitly identified in the chapter that makes the claim. The reader can verify, for any specific claim, which types support it and where the claim's

structural support ends. The framework's universality claim survives precisely because its claims are scoped to the regimes where the supporting structure holds.

**What the principle does not promise.** The Sudoku Principle is not an algorithm for generating new mathematical theorems. It is an epistemic discipline that applies after the structural reading has been done. The structural reading itself—the recognition that a chain ring and a lawnmower blade are both $\mathbb{S}^1$, that a spectrum's mass gap is a curvature tax, that a cross-correlation block is a holonomy—requires the kind of recognition the framework attributes to the kid in the driveway. The discipline catches the recognition's failure modes; it does not produce the recognition.

The framework's recognitions came from many specific instances over a long compounding period—the lawnmower afternoon, the bra strap morning, the mass gap night, the hawk-in-a-turn observation, Teddy's ears at the kitchen table, the kid heading across town. The Sudoku Principle is the discipline that lets those recognitions be carried forward as load-bearing claims rather than as suggestive analogies.

§ 23.8

## *What you take from this chapter*

Four things.

**First.** The Sudoku move is one question: *if this is true, what else follows?* The structure does the work; the solver reads the structure honestly. The move is mechanical, not magical, and its discipline is the refusal to add what the structure does not entail.

**Second.** The framework's *epistemic discipline for marking support* is a catalog of four convergence types: local-to-global propagation (the Bra Strap Principle); cross-domain reading (the Davis Field Equation's universality); limit convergence to classical (recovering Taylor, LQR, Kalman as flat-limit special cases); and multi-method agreement (independent solvers converging). A claim is carried by the combination of types that support it, with the support marked explicitly. None of the four is a proof in the mathematical sense—each is evidence of a specific structural kind—and the framework treats them as such.

**Third.** The framework consistently distinguishes structural readings (the object plays the role) from literal identifications (the object is the role). Where a literal identification is available, the framework states it; where only a structural reading is available, the framework states that and marks the precise identification as open. The book's softenings across Chapters 7, 12, 16, and 21 are applications of this discipline.

**Fourth.** The Sudoku Principle (§23.7) is the formal statement of the discipline. It commits the framework to marking each claim's support accurately, identifying overreach and undermarking as the named failure modes, and naming the open work where structural readings await literal identification. The framework's universality claim is carried by the discipline, not by the strength of the rhetoric.

Chapter 25 closes the book by returning to the kid in the driveway, naming what's still open in the Davis-Geometric program, and pointing to the work the framework continues to do beyond this volume.

*Vamos.*

Chapter 25

# *The kid, looking back; the framework, looking forward*

*The work of every individual must necessarily limit itself to a single fragment of the immense whole, but the work of all together is precisely the work of the immense whole.*
*Henri Poincaré, on collective mathematical progress, paraphrased from Science and Method (1908)*

§ 24.0

## *The kid is grown*

Twenty-four chapters and six labs ago, a kid in a driveway looked at a lawnmower engine and a BMX bike and recognized that they had the same geometry. She did not have the words *fiber bundle* or *section* or *associated principal bundle*. She had the recognition. The recognition told her to disassemble the lawnmower, to lay the parts on the driveway, to bend a bracket, to thread a chain. The same recognition told her, before she rode, to size the recovery

authority of the brakes. The bike had brakes that could stop a BMX frame propelled by a child's legs; the bike with the engine bolted on would have the kinetic energy of an internal-combustion-driven projectile. The brakes did not match the energy. The kid did not ride.

She is the protagonist of this book. She is also the author.

The recognition the kid had on the driveway is the recognition the framework has been formalizing for twenty-four chapters and six labs. The Bra Strap Principle is the formal statement of how a chain ring and a lawnmower blade can be the same. The Davis Field Equation is the formal statement of when a curved manifold's local approximation must charge a curvature tax. The Substrate Principle is the formal statement of how a controller can read a single scalar and trust it. The Sudoku Principle is the formal statement of how the framework's claims are carried by the structures that support them, not by the strength of the language used.

The kid, formalized.

That has been the book's commitment from Chapter 1's opening: *the math is the kid, formalized.* By Chapter 25, twenty-four chapters of formalization later, the commitment is what the book has paid out.

§ 24.1

## *What the book has done*

The book has walked the GNC stack through the framework's geometric vocabulary:

**Foundations** (Chapters 1–3). The Bra Strap Principle, the Davis Field Equation, the geometric vocabulary the rest of the book uses without reintroduction.

**Plant** (Chapters 5–6). Birds in flight, quadcopter bundles, the F-16 longitudinal model. The plant read as a coupled fiber bundle whose structure the controller acts on.

**Control** (Chapters 7–9, Lab 9). The Bode integral as a structural constraint, LQR as gradient descent on the value function, $H_\infty$ as worst-case shaping in the same geometric language.

**Information and adaptation** (Chapters 10–13, Lab 12). The Kalman filter on the Fisher manifold, Lyapunov's method as geometric stability, MRAC as adaptation on the parameter bundle, the Curvature Floor conjecture as the framework's central inequality.

**Navigation** (Chapters 14–16, Lab 16). Inertial mechanization as parallel transport, GNSS/INS integration as a coproduct of bundles, SLAM as the same coproduct extended with the map summand.

**Architecture** (Chapters 17–20, Lab 20). The Substrate Principle, GIGI as the bundle database, DHOOM as the bundle-aware serialization, the integrated ICARUS stack with its defensive commitment enforced at the type-system level.

**Guidance** (Chapters 22–23, Lab 23). Trajectories as geodesics on the cost manifold, the four classical guidance laws as four cost-functional instances of the same machinery.

**Methodology** (Chapter 24). The Sudoku Principle, the framework's epistemic discipline named formally.

Eight parts. Twenty-four chapters. Six labs. One framework. The book's universality claim is that the same geometric machinery—bundles, sections, connections, curvature, the Davis Field Equation—reads each of these domains, and the reading produces consistent answers across the domains. The reader who has run the labs has empirical evidence for the consistency on small instances; the reader who has read the chapters has the structural argument for the consistency in general.

§ 24.2

## *What's still open*

Honesty about what's open is part of the discipline the previous chapter named.

**The literal differential-geometric formulations.** Four places in the book carry structural readings that await literal mathematical identifications: the Bode integral as Gauss-Bonnet on the controller bundle (Chapter 7 §7.4); persistency of excitation as a holonomy condition on the regressor bundle (Chapter 12 §12.8); the SLAM cross-correlation block as the connection's holonomy (Chapter 16 §16.2); and the Pontryagin costate as the cotangent transport object on the cost manifold (Chapter 22 §21.4). In each case, the structural reading is consistent and operationally useful; the precise differential-geometric statement, with the connection specified and the identification proved, is open work for the geometric-control and applied-topology literatures to ratify.

**The Davis Duality's multiplicative constant.** The Davis Duality's two-sided bound is established (three proof routes in the 2026 Zenodo paper). The constant in front of the bound,

for general cost functionals, is not. The constant is what would convert the framework's structural claim into a numerical Davis Duality lower bound; deriving the constant is the next theorem the framework needs.

**The Curvature Floor conjecture.** Chapter 13's inequality form remains a conjecture in the nonlinear adaptive regime. The LTI special case (Chapter 7 §7.5) is proved by the standard Bode argument. The full nonlinear-adaptive case requires the Davis Duality bound's constant and a Lyapunov-monitor argument that extends to time-varying regressor bundles. Both pieces are open work.

**The Yang-Mills mass-gap paper.** The mass-gap manuscript (Davis 2025, Zenodo DOI `10.5281/zenodo.17846521`) is a structural reading of the gap as one instance of the Davis Field Equation, with the $15\sigma$ empirical confirmation from the lattice computation. Whether the structural reading constitutes a contribution to the Clay Millennium Prize evaluation is a question for the expert mass-gap community to settle through the standard Clay process. The book's claim depends on the structural reading, not on the Clay verdict.

**The other domains.** The framework's reading of Navier-Stokes regularity, P-versus-NP, Poincaré (Perelman's proof), and the Riemann hypothesis are structural consistency checks, not proofs. Each is reported in the Davis-Wilson Lattice repository's documentation, scoped to its own specific consistency check, and held to the framework's epistemic discipline about what the structural reading does and does not entail.

**The Marcella conjectures.** The Dual-Bundle Theorem (Marcella, §21.5) is proved under its nondegeneracy hypothesis. Three of its four named consequences remain open: the Qualitative Topological Obstruction via Chern-Weil characteristic classes (rigorous corollary — carried), the Quantitative Hodge Lower

Bound (conjecture), and "cross-entropy as integrated holonomy discrepancy" (conjectural bridge, supported empirically). The hypothesis from §21.4 that explicit holonomy is the mechanism for compositional residue in language is an architectural hypothesis with no theorem behind it; its falsification would not undermine the rest of the framework but would re-scope Marcella's claims.

**Specific framework extensions named in the chapters.** The Hopf-map / cascade-bandwidth / saturation interaction in the quadcopter coupling (§5.7) has been reframed away from a literal Hopf-degeneracy claim, but the formal account of how cascade-bandwidth deficit, sign discontinuity, and saturation combine into a single curvature obstruction is unfinished. The aerodynamic bundle of Chapter 5 is described against incompressible Navier-Stokes; the compressible-flow extension (transonic, supersonic) is left as work. The cost-manifold geodesic of Chapter 22 is most naturally a sub-Riemannian / sub-Finsler object (Appendix C); the Davis Duality bound's extension to non-Riemannian cost manifolds is named but not derived.

**What this list is for.** The framework's universality claim does not depend on any one of these open items closing. Each is named here so that the reader who wants to contribute to the program has a punch-list, and so that no claim in the book trades on something the framework has not actually proven. The Sudoku Principle of Chapter 24 carries the discipline; this section is its application to the book's own contents.

§ 24.3

## *What comes next*

The Davis-Geometric program continues beyond this volume. The series so far:

***Hidden Variable: Unlocking Patterns in a World Obsessed with Structure*** (Davis, 2025). The diagnosis-of-exclusion that structured the author's early life, and the pattern-recognition methodology the author built in response. The autobiographical book that makes *The Geometry of Flight*'s "kid in the driveway" a concrete person.

***The Geometry of Sameness: Riemannian Equivalence of Translation and Distance for Semantic Detection*** (Davis, December 2025). The Book 2 in the *Geometric Intelligence* subseries. Develops the categories-of-realizations and Error Budget Transfer Theorem the present book uses operationally.

***The Bra Strap Principle: Flat Base Propagation in Fiber Bundles over*** $\mathbb{S}^1$ (Davis, 2026). The formal monograph stating the propagation theorem the framework uses across every chapter of the present book.

***The Davis Duality of Approximation and Obstruction: Why Machine Learning Works, Why the Vacuum Has Mass, and the Universal Law of Flat Failure*** (Davis, 2026). The formal monograph stating the two-sided bound the framework uses operationally as the Davis Field Equation in flat-tangent regimes.

**Forthcoming.** The author's working agenda includes a volume on the framework's read of energy and thermodynamics (*The Geometry of Fuel*); a volume on bilateral diplomacy as helicity on coupled relationship spaces (*The Geometry of War*); a volume on phase transitions and criticality (*Superfluid*); a volume on the

framework's read of cognition and language (the next *Geometric Intelligence* book, building on *Sameness*); and the formal write-ups of the Yang-Mills, Navier-Stokes, P-versus-NP, Poincaré, and Riemann consistency checks as separate domain-specific manuscripts. The pace at which these arrive will depend on the world that the framework is being built into, and the readers who pick up the work and carry it forward.

**The operational stack.** ICARUS, GIGI, DHOOM, and the supporting tools (MIRADOR, PRISM, MARCELLA, DEMETER) continue to develop. The patent record stands at three provisionals as of this writing (ICARUS, GIGI, DHOOM, all filed April 21, 2026 and March 18, 2026 respectively). Source code is in the Davis-Geometric organizational repositories; the open-source components (DHOOM in particular) are installable from the standard package indices.

§ 24.4

## *The reader's role*

A book is only an artifact of the work that produced it. The work continues after the book is published, and the reader is part of how the work continues.

**What the reader who got here can do.** The labs in this book are runnable on any modern laptop. They are demonstrations and sanity checks on the framework's specific instances, not verifications of the universality claim itself (which is the argument the chapters make, with the convergence types of §24.2 marking the support). The reader who runs Lab IV and watches $\widehat{\kappa}$ drop at loop closure has produced one positive instance of the framework's prediction at the lab scale; multiplying instances of the

same kind across domains is what the cross-domain convergence type accumulates. That is a contribution.

**What the reader who builds with the framework can do.** The bundle vocabulary, the Substrate Principle, and the Sudoku discipline are not proprietary. The framework's claim is that the vocabulary reads many domains, and the reader who applies it to a domain the book has not touched is producing new structural readings the framework would want to know about. The author's contact information is in the bibliography; the standard scholarly channels (Zenodo, arXiv, the conference circuits) are also routes.

**What the reader who finds an error can do.** The book has been written with the epistemic discipline of Chapter 24 applied as carefully as the author could apply it. Errors will be present anyway. A reader who finds a structural reading that overreaches, an undermarked design choice, a missing constraint, or a numerical claim that does not match the lab's actual output is reading the book correctly and is invited to file a correction. The book is not the final word on the framework; the labs and the source code are checks against which the book must continue to hold.

**What the reader who disagrees can do.** The framework's claims are scoped to specific configurations and supported by specific convergence types. A reader who finds the support insufficient, or who finds a configuration where the support fails, is making the framework better by saying so. The book's commitment is to the discipline of Chapter 24, not to the immutability of any specific claim within the book.

§ 24.5

## *A final return to the driveway*

The kid in the driveway is twelve years old. Maybe thirteen. Her mother has gone inside. The lawnmower's parts are on the driveway in the order she removed them. The bracket she bent to hold the engine to the bike frame is sitting on the workbench. The chain she pulled from somewhere is coiled at her feet. The bike is on its kickstand. The sun is angling down through the carport.

She runs the trajectory forward in her head. The engine starts. The chain engages. The bike accelerates. The front wheel lifts. The street ends at a stop sign. She reaches for the brakes.

*No tenía los frenos.*

She did not have the brakes. The brake pads were sized for a child's bike. The engine had added the kinetic energy of a small motorcycle. The kid had the actuator. The kid had the coupling. The kid had the recovery margin only of the bike she was about to abolish.

She put the parts back. She made herself a sandwich. She decided that day, without having the words for it, that she would not build a thing that did not include its own recovery authority.

That decision became this book.

The book has been a long way of saying what the kid already knew on the driveway. The geometric vocabulary, the bundle constructions, the curvature-gated modulation, the substrate principle, the type-system commitments, the Sudoku discipline, the labs that verify the claims on actual code—they are the formal version of the kid's recognition. Every chapter, in some way, is a derivation from that afternoon.

The kid did not ride. The framework she went on to build does not ride either, in the sense that ICARUS is a defensive

architecture committed at the schema layer not to be weaponized. The kid's recognition and the framework's commitment are the same recognition: *do not build a thing that does not include its own recovery authority.* The book has unpacked that recognition across twenty-four chapters. The recognition itself is older than the book.

§ 24.6

## *Vamos*

The Spanish word the book has used at moments of recognition, of structural identification, of the framework recognizing itself in the world: *vamos.* It means *let us go.*

The book closes here, but the recognition continues. The kid is grown. The framework is built. The substrate runs. The labs are reproducible. The patents are filed. The papers are on Zenodo with DOIs that will outlive the author. The series continues. The reader, having read this book, is now part of the system that carries the recognition forward.

Everything the book has said is a structural reading of a single moment in a driveway when a twelve-year-old recognized that she did not have the brakes, and decided not to ride.

Everything that comes next is the consequence of that decision, carried forward into the world by every reader who picks up the work and asks, of whatever domain they are in: *if this is true, what else follows?*

*Vamos.*

Chapter A

# *Notation index*

A reader who opens the book at the middle will see symbols whose meaning was fixed many pages earlier. This appendix lists each symbol the book uses, with the section where it is first defined and a short gloss. Symbols whose meaning is purely local to one equation are omitted.

## Framework scalars and operational quantities

$C$ Operational confidence. Dimensionless, in $(0, 1]$. Defined as $C = \tau/K$ (scaling form, §2.0) or $C = 1/(1+\hat{\kappa})$ (operational form, §2.3).

$\tau$ Tolerance. Dimensionless ratio of admissible error to characteristic error magnitude (§2.1).

$K$ Curvature obstruction (un-normalized). Has units of inverse length, or its domain-specific analogue (§2.1).

$\hat{\kappa}$ Dimensionless, non-negative curvature obstruction (§2.3). Defined as $\hat{\kappa} = \ell^2 \|\Omega\|_g$ for a characteristic length $\ell$; the framework's load-bearing operational scalar.

$\ell$ Characteristic length scale; control-cycle length or local lattice spacing (§2.3).

$m(C)$ Gain modulation schedule; $m(C) = m_{\min} + (m_{\max} - m_{\min}) \cdot C^{\gamma}$ with $m_{\min}, m_{\max} \in (0, 1]$ and $\gamma > 0$ (§8.6).

### Bundles, sections, connections

$E \to B$ A fiber bundle over base $B$ with total space $E$. $\pi : E \to B$ is the projection (§3.1).

$\sigma$ **(or** $s$**)** A section of a bundle (§3.2).

$\nabla$ A connection (covariant derivative) on a bundle (§3.2).

$\omega$ A connection 1-form (§3.2).

$\Omega$ The curvature 2-form of a connection; $\Omega = d\omega + \omega \wedge \omega$ (§3.3).

$\mathrm{Hol}_p(\omega)$ The holonomy group at $p$ of the connection $\omega$ (§3.3).

$P_\gamma$ Parallel transport along the loop $\gamma$ (§3.2).

### Attitude and rotations

$\mathrm{SO}(3)$ The group of rotations of $\mathbb{R}^3$. Three-dimensional, $\pi_1(\mathrm{SO}(3)) = \mathbb{Z}/2$ (§3.4).

$S^3$ The 3-sphere in $\mathbb{R}^4$; the unit quaternions; the universal (double) cover of $\mathrm{SO}(3)$ (§3.4).

$q = (q_0, q_1, q_2, q_3)$ A unit quaternion, $|q| = 1$ (§3.4).

$q_{\mathrm{err}} = q_{\mathrm{cmd}}^{-1} \otimes q$ Multiplicative attitude error; with short-arc sign correction this is the standard error signal for quaternion control (§3.4, §5.7).

$\hat{z}_b^{\mathrm{world}}$ Third column of $\mathbf{R}(q)$; the body-$Z$ axis expressed in the world frame (§5.5).

### Control and estimation

$\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ State-space matrices of a linear plant: $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$, $\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$ (§6).

$\mathbf{K}_{\mathrm{lqr}}$ The LQR feedback gain; $\mathbf{K}_{\mathrm{lqr}} = \mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}$ (§8.1).

**$\mathbf{P}$ (LQR)** Solution of the algebraic Riccati equation; Hessian of the optimal value function (§8.2).

**$\mathbf{P}$ (Kalman)** Posterior state covariance of the Kalman filter (§10.1). *Two distinct uses of the same symbol; the context disambiguates.*

**$\mathbf{Q}, \mathbf{R}$** State-cost and control-cost matrices for LQR (§8.1); process-noise and measurement-noise covariances for the Kalman filter (§10.1). Same symbols, different roles—again context-dependent.

$L(s)$ Open-loop transfer function $L(s) = P(s)K(s)$ (§7.1).

$S(s) = 1/(1 + L(s))$ Closed-loop sensitivity function (§7.1).

$T(s) = L(s)/(1 + L(s))$ Complementary sensitivity (§7.1).

$M_S = \|S\|_\infty$ Sensitivity peak; the worst-case closed-loop amplification of output disturbance (§9.1).

$\|G\|_\infty = \sup_\omega \bar{\sigma}(G(j\omega))$ $H_\infty$ norm of $G(s)$ (§9.1).

## Adaptive control

$\boldsymbol{\theta}$ Unknown parameter vector ($\boldsymbol{\theta} \in \mathbb{R}^d$) (§11.4).

$\hat{\boldsymbol{\theta}}$ Online estimate of $\boldsymbol{\theta}$ (§11.4).

$\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} - \hat{\boldsymbol{\theta}}$ Parameter error (§11.4).

$\boldsymbol{\Gamma}$ Adaptive-law gain matrix ($\boldsymbol{\Gamma} \succ 0$, $d \times d$) (§11.4). *Distinct from the Christoffel symbol $\Gamma^i_{jk}$ in §21.2 and from the composite transport operator $\Gamma$ in Marcella §21.3.*

$\mathbf{Y}(\mathbf{x})$ Matrix-valued regressor, $\mathbf{Y}(\mathbf{x}) \in \mathbb{R}^{m \times d}$, with $\mathbf{Y}(\mathbf{x})\,\boldsymbol{\theta} \in \mathbb{R}^m$ matching $\mathbf{u}$'s dimension (§11.4, §12.2).

$\boldsymbol{\phi}(\mathbf{x})$ SISO regressor vector, $\boldsymbol{\phi} \in \mathbb{R}^d$; the row of $\mathbf{Y}$ in the SISO case (§11.4).

$\mathbf{A}_m, \mathbf{B}_m$ Reference-model state and input matrices (§12.2).

$\mathbf{e} = \mathbf{x} - \mathbf{x}_m$ Tracking error (§12.3).

## Navigation, fusion, SLAM

**$\boldsymbol{\omega}$ (body frame)** Angular velocity in body coordinates (§5.1, §14.2).

$\mathbf{f}_{\text{spec}}$ Specific force from accelerometer, body frame (§14.3).

$\mathbf{I}_{\text{fused}} = \mathbf{P}_{\text{fused}}^{-1}$ Fused Fisher information; sum of independent sensors' information matrices in the linear-Gaussian limit (§15.4).

$\mathbf{m}_i$ Landmark position in SLAM (§16.1).

$\boldsymbol{\Sigma}_w$ Whitened joint covariance for SLAM (§16.5).

## Special objects in late chapters

$g_{\mathbf{P}}$ LQR-induced Riemannian metric on state space: $g_{\mathbf{P}}(\mathbf{x}, \mathbf{y}) = \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{y}$ (§8.5).

$\Gamma^i_{jk}$ Christoffel symbols of the cost-manifold metric (§21.2).

$\Omega_\Delta = \Omega_D - \Phi^*\Omega_M$ Mismatch curvature between data and model bundles in Marcella (§21.5).

$\mathrm{Hol}_\gamma^{\nabla_D}$, $\mathrm{Hol}_\gamma^{\Phi^*\nabla_M}$ Holonomies of the data and pulled-back model connections around loop $\gamma$ (§21.5).

## Operators and conventions

$\otimes$ Quaternion product (§3.4, §14.2).

$\succ 0$, $\succeq 0$ Positive (semi)definite under the Loewner partial order.

$\|\cdot\|$ Euclidean norm by default; subscripted norms ($\|\cdot\|_\infty$, $\|\cdot\|_g$, $\|\cdot\|_F$) named where they appear.

$\mathrm{tr}(\mathbf{P})$ Trace of $\mathbf{P}$ (gauge-dependent unless the coordinates are pre-whitened; see §10.4 caveat).

$\bar{\sigma}(\cdot)$ Largest singular value (§9.1).

$\sup_\omega$ Supremum over real frequencies $\omega$ (§9.1).

Chapter B

# *Framework glossary*

The Davis-Geometric framework names a number of principles, theorems, and engineering objects. This appendix collects them in one place, with one-paragraph definitions and pointers to where each is developed in the book.

**Bra Strap Principle.** A local check on a fiber bundle's base, applied at one point on a connected region where the connection is flat (or, more generally, on a region where the required propagation conditions hold), determines the section globally on that region. Where the conditions fail, the local check determines the section up to the connection's holonomy. Davis 2026, Zenodo DOI 10.5281/zenodo.20273573. See Chapters 1 (informal) and 3 (precise statement via Ambrose–Singer).

**Curvature Floor (conjecture).** The adaptive extension of the Bode sensitivity integral. For a closed loop with an adaptive law, the integrated $\log|S(j\omega)|$ is bounded below by the classical Bode term plus an adaptive-cost term scaling with the inverse gain matrix $\mathrm{tr}(\mathbf{\Gamma}^{-1})$ plus a parameter-manifold-curvature term. Conjectural; the empirical monotonicity has been measured. Chapter 13.

**Davis Duality of Approximation and Obstruction.** A two-sided bound: the error of a flat-tangent approximation

against a curved section is bounded above *and below* by a multiple of the integrated curvature times the step size squared. The lower bound is the news: there is a structural error floor that no refinement of the flat approximation can get below. Davis 2026, Zenodo DOI 10.5281/zenodo.19428406. Used throughout; introduced in Chapter 1.

**Davis Field Equation.** $C = \tau/K$ in its scaling form, $C = 1/(1+\hat{\kappa})$ in its operational form. The framework's central operational relation between confidence, the tolerance budget, and the curvature obstruction. Chapter 2.

**DHOOM.** The serialization layer of the bundle substrate; preserves geometric type on the wire. U.S. Provisional Patent 64/008,940 (March 18, 2026). Chapter 19.

**Dual-Bundle Theorem (Holonomy Sandwich).** An extension of the Davis Duality to two bundles (data and model). Under nondegeneracy on the loop's orientation and curvature mismatch, the discrepancy between the two bundles' holonomies on small loops is sandwiched by the mismatch curvature times the loop area. Used in Marcella's geometric account of training (Chapter 21, §21.5).

**GIGI.** The fiber-bundle database engine that stores bundles natively, with a connection as a first-class object and sheaf completion as a primary operation. U.S. Provisional Patent 64/045,889. Chapter 18.

**Holonomy-gated control.** The architectural commitment that the controller's effective gain is modulated by an estimate of local holonomy along the predicted trajectory. Operationally realized as $m(\hat{\kappa}) \cdot \mathbf{K}_{\text{lqr}}$. Chapter 8 §8.6,

Chapter 20 §20.1.

**ICARUS.** The flight-control architecture that emerged from the framework. Five layers: sensors/actuators, the GIGI substrate, DHOOM serialization, controller-and-supervisor, mission and ground. U.S. Provisional Patent 64/045,891 (April 21, 2026). Chapter 20.

**MARCELLA.** The framework's transformer-replacement architecture for language. Reads attention as parallel transport on a fiber bundle whose base is the token-position interval and whose fibers are semantic (on $\mathbb{S}^7$) and grammatical. Chapter 21 (Marcella file).

**Sentinel.** The non-actuating defensive-perception module of ICARUS. Emits proximity and risk advisories; the architecture's type system enforces that the sentinel cannot write to actuator sections. Chapter 20 §20.3–20.4.

**Substrate Principle.** The architectural commitment that all layers communicate through a single narrow interface (GIGI), with no direct cross-layer coupling. The Sudoku Principle's architectural realization. Chapter 17.

**Sudoku Principle.** The framework's epistemic discipline for marking the support behind each claim, using a catalog of four convergence types: local-to-global propagation, cross-domain reading, limit convergence to classical, and multi-method agreement. None of the four is a mathematical proof; each is a kind of evidence. Chapter 24.

Chapter C

# *Sub-Riemannian and sub-Finsler geometry, briefly*

Chapter 23 §22.4 invokes *sub-Riemannian* and *sub-Finsler* structure for minimum-time and minimum-fuel optimal control. This appendix gives the minimum vocabulary needed to read that passage, with no claim of completeness.

**What's sub- about it.** Ordinary Riemannian geometry assigns a length to every smooth curve via an inner product on the tangent space: at each point $p \in M$, the tangent space $T_pM$ carries a positive-definite metric $g_p$, and the length of a curve $\gamma$ is $\int \sqrt{g_{\gamma(t)}(\dot\gamma, \dot\gamma)}\, dt$. The minimum-length curve between two points is a geodesic.

A *sub-Riemannian* manifold restricts which directions are "admissible" at each point. Concretely, at each $p \in M$, you specify a subspace $D_p \subseteq T_pM$ (the *horizontal distribution*), and only curves whose velocity lies in $D_p$ at every point are admissible. The metric is defined on $D_p$ only; directions transverse to $D_p$ have no defined length and cannot be traveled directly.

The classical example is the *Heisenberg group*: a 3-manifold with a 2-dimensional horizontal distribution at each point. You can move in two directions instantaneously, but to move in the third you have to "corkscrew"—trace a small loop in the first two directions, and the third direction emerges as the loop's enclosed area. Sub-Riemannian distance to a transverse point is positive

but not zero, and shortest paths are not what you would draw in Euclidean coordinates.

**Why this matches minimum-time control.** Consider a plant $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ with $\mathbf{u} \in U$ admissible. At each state $\mathbf{x}$, the set of velocities $\dot{\mathbf{x}}$ reachable by some admissible $\mathbf{u}$ is a subset of $T_{\mathbf{x}}M$—the *reachable velocity cone*. For minimum-time problems, this is the natural horizontal distribution, and the minimum time between two states is the sub-Riemannian (or sub-Finsler, if the unit-velocity set is not an ellipsoid) distance. Bang-bang controls correspond to geodesics that ride the boundary of the reachable velocity cone almost everywhere; singular arcs correspond to geodesics tangent to interior structure of the cone. The Pontryagin necessary conditions of Appendix E are the sub-Riemannian geodesic equation in this setting.

**Why this matches minimum-fuel control.** An $L^p$ cost on the control input, $\int \|\mathbf{u}\|_p^p \, dt$, induces a *Finsler* norm on the reachable velocity cone—a norm that is not necessarily quadratic. The minimum-cost trajectory is a Finsler geodesic. Cases:

- $p = 2$ (quadratic effort): the Finsler norm is the sub-Riemannian metric the control allocation induces; geodesics are smooth, controls are smooth.
- $p = 1$ (fuel cost, $L^1$ effort): the unit-cost set has corners (it is a polytope when $\mathbf{u}$ ranges over a box). Geodesics are bang-off-bang with singular arcs; controls are not in general smooth.
- $p = \infty$ (peak-effort cost): the unit-cost set is the reachable velocity polytope's boundary in the max-norm sense; geodesics are again bang-bang on each input channel.

**What the book uses this for.** Chapter 22 reads guidance as geodesic computation on a cost manifold; the cost functional's

choice determines whether the right structure is sub-Riemannian (minimum time, $L^2$ effort) or Finsler (general $L^p$ effort, fuel, peak constraints). The Davis Duality of approximation and obstruction applies on the configuration manifold itself; the sub-Riemannian / Finsler refinement is what makes the cost-manifold geodesic the actual optimal control rather than a Riemannian-metric straw man.

**Where to read more.** The standard introduction is Montgomery, *A Tour of Sub-Riemannian Geometries, Their Geodesics and Applications* (AMS, 2002). For the connection to geometric control, Agrachev and Sachkov, *Control Theory from the Geometric Viewpoint* (Springer, 2004).

Chapter D

# *Fisher information and natural gradient, briefly*

Chapter 10 invokes Fisher information, the Fisher metric, the natural gradient, and the Cramér-Rao bound. This appendix gives the minimum vocabulary needed, with the linear-Gaussian special case worked through.

**The Fisher information matrix.** Let $p(\mathbf{z} \mid \boldsymbol{\theta})$ be a probability density parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$. The *Fisher information matrix* at $\boldsymbol{\theta}$ is

$$\mathbf{I}(\boldsymbol{\theta}) \;=\; \mathbb{E}_{\mathbf{z}|\boldsymbol{\theta}}\Big[\, \nabla_{\boldsymbol{\theta}} \log p(\mathbf{z} \mid \boldsymbol{\theta})\; \nabla_{\boldsymbol{\theta}} \log p(\mathbf{z} \mid \boldsymbol{\theta})^{\mathrm{T}} \,\Big].$$

Equivalently, $\mathbf{I}(\boldsymbol{\theta}) = -\mathbb{E}\big[\nabla^2 \log p\big]$ (negative expected Hessian) under mild regularity. The matrix is positive semidefinite, and positive definite when the parameterization is locally identifiable.

**What it measures.** $\mathbf{I}(\boldsymbol{\theta})$ is the local curvature of the log-likelihood surface as a function of $\boldsymbol{\theta}$: directions in which $\mathbf{I}$ is large are directions in which the data is informative; directions in which $\mathbf{I}$ is small are directions in which the data is ambiguous.

**The Cramér-Rao bound.** For any unbiased estimator $\hat{\boldsymbol{\theta}}(\mathbf{z})$ of $\boldsymbol{\theta}$,

$$\mathrm{Cov}(\hat{\boldsymbol{\theta}}) \;\succeq\; \mathbf{I}(\boldsymbol{\theta})^{-1}.$$

The Fisher information's inverse is a lower bound on the achiev-

able estimation covariance. Estimators that achieve this bound are *efficient*. The Kalman filter achieves it in the linear-Gaussian case.

**The Fisher metric.** $\mathbf{I}(\boldsymbol{\theta})$ defines a Riemannian metric on the parameter manifold: the inner product on $T_{\boldsymbol{\theta}}\Theta$ is $\langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{I}} = \mathbf{u}^{\mathrm{T}} \mathbf{I}(\boldsymbol{\theta}) \mathbf{v}$. The resulting Riemannian geometry on $\Theta$ is the subject of *information geometry* (Amari and Nagaoka, *Methods of Information Geometry*, AMS, 2000; Amari, *Information Geometry and Its Applications*, Springer, 2016).

**The natural gradient.** An ordinary gradient $\nabla_{\boldsymbol{\theta}} L$ of a loss $L$ is a coordinate- dependent quantity; rotating $\boldsymbol{\theta}$ rotates the gradient differently than the loss landscape. The *natural gradient*, defined by

$$\tilde{\nabla} L \;=\; \mathbf{I}(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} L,$$

is coordinate-invariant in the Fisher metric—a steepest-descent direction with respect to the Fisher metric rather than the Euclidean metric on coordinates. For probabilistic models the natural gradient is the principled descent direction. For Gaussian likelihoods with parameter the posterior mean, the natural gradient update is exactly the Kalman filter's measurement update, up to a choice of step size that the Kalman gain selects optimally.

**The linear-Gaussian special case.** For $\mathbf{z} \sim \mathcal{N}(\mathbf{H}\boldsymbol{\theta}, \mathbf{R})$,

$$\mathbf{I}(\boldsymbol{\theta}) \;=\; \mathbf{H}^{\mathrm{T}} \mathbf{R}^{-1} \mathbf{H}.$$

Two independent Gaussian observations of the same $\boldsymbol{\theta}$ produce additive Fisher information matrices: $\mathbf{I}_{\text{fused}} = \mathbf{I}_1 + \mathbf{I}_2$. This is the information-filter form of the Kalman update and the foundation of the sensor-fusion identity $\mathbf{P}_{\text{fused}}^{-1} = \mathbf{P}_1^{-1} + \mathbf{P}_2^{-1}$ used in Chapter 15.

**What the book uses this for.** Chapter 10 reads the Kalman

filter as the natural-gradient estimator on the Fisher manifold in the linear-Gaussian limit. The framework's operational scalar $\hat{\kappa}$ for sensor fusion is defined as a non-negative function of $\mathrm{tr}(\mathbf{P})$—a design choice motivated by the Fisher-geometric intuition, but not literally the Fisher-Ricci scalar curvature (which is signed, parameterization- dependent, and expensive to compute).

Chapter E

# *Pontryagin's Minimum Principle, briefly*

Chapters 22 and 23 invoke Pontryagin's principle without developing it. This appendix gives the statement, the necessary conditions, and the connection to the cost-manifold geodesic reading the book uses.

**The optimal control problem.** Given dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$ with state $\mathbf{x} \in \mathbb{R}^n$ and control $\mathbf{u} \in U \subseteq \mathbb{R}^m$, and a cost functional

$$J(\mathbf{u}) \;=\; \phi(\mathbf{x}(t_f), t_f) \;+\; \int_{t_0}^{t_f} L(\mathbf{x}, \mathbf{u}, t)\, dt,$$

find the control $\mathbf{u}^*(t)$ on $[t_0, t_f]$ that minimizes $J$ subject to the dynamics and any state or control constraints.

**The Hamiltonian.** Introduce the *costate* (adjoint state) $\boldsymbol{\lambda}(t) \in \mathbb{R}^n$. Define the Hamiltonian

$$H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, t) \;=\; L(\mathbf{x}, \mathbf{u}, t) \;+\; \boldsymbol{\lambda}^{\mathrm{T}} \mathbf{f}(\mathbf{x}, \mathbf{u}, t).$$

**Pontryagin's necessary conditions.** If $\mathbf{u}^*(t)$ is optimal and $\mathbf{x}^*(t)$ is the corresponding state trajectory, there exists a costate $\boldsymbol{\lambda}^*(t)$ such that:

1. *State equation:* $\dot{\mathbf{x}}^* = \partial H / \partial \boldsymbol{\lambda} = \mathbf{f}(\mathbf{x}^*, \mathbf{u}^*, t)$.
2. *Costate equation:* $\dot{\boldsymbol{\lambda}}^* = -\partial H / \partial \mathbf{x}$ (evaluated along the

optimal trajectory).

3. *Minimum condition:* for each $t$, the optimal control minimizes the Hamiltonian over the admissible set:

$$\mathbf{u}^*(t) \;=\; \arg\min_{\mathbf{u}\in U} H\big(\mathbf{x}^*(t), \mathbf{u}, \boldsymbol{\lambda}^*(t), t\big).$$

4. *Transversality:* $\boldsymbol{\lambda}^*(t_f) = \partial\phi/\partial\mathbf{x}$ (or boundary-condition-specific variant).

The minimum condition over the admissible set $U$ is what gives bang-bang controls when $H$ is linear in $\mathbf{u}$ and $U$ is a bounded set: the minimum of a linear function over a polytope occurs at a vertex of the polytope, and $\mathbf{u}^*$ switches between vertices as $\boldsymbol{\lambda}^*$ evolves.

**The costate as cotangent.** The geometric reading of the costate is that $\boldsymbol{\lambda}^*(t)$ is a cotangent vector along the optimal trajectory: at each point of $\mathbf{x}^*$, $\boldsymbol{\lambda}^*$ lives in $T^*_{\mathbf{x}^*}M$ and encodes "the sensitivity of the optimal cost-to-go to a perturbation of the current state." The costate equation is the parallel-transport equation for cotangent vectors along $\mathbf{x}^*$ under the connection that the dynamics $\mathbf{f}$ and the cost $L$ jointly induce.

This is the geometric content the book uses: Pontryagin's principle is the cotangent-bundle parallel-transport equation on the cost manifold; the necessary conditions are the geodesic equation written in cotangent coordinates rather than in tangent coordinates. The principle's Hamiltonian-flow formulation in symplectic geometry makes this explicit; this primer omits the symplectic structure.

**The cost-manifold reading.** In Chapter 22 §21.2, the optimal trajectory is read as a geodesic on the cost manifold equipped with a Riemannian (or sub-Riemannian / Finsler) metric induced by the cost functional. Pontryagin's principle is equivalent to the geodesic equation in this metric; the costate $\boldsymbol{\lambda}^*$ is the dual vector

(the metric's musical-isomorphism image) along the geodesic. The framework's $\hat{\kappa}$ on the cost manifold reads the curvature of the geodesic equation and tells the planner when the LQR-flat approximation is structurally inadequate.

**Where to read more.** The classical treatment is Bryson and Ho, *Applied Optimal Control* (Blaisdell, 1969). The geometric formulation is in Agrachev and Sachkov, *Control Theory from the Geometric Viewpoint* (Springer, 2004), and Bullo and Lewis, *Geometric Control of Mechanical Systems* (Springer, 2005).

Chapter F

# *End-to-end worked example: one X4 control cycle*

This appendix walks one millisecond of an X4 quadcopter flight through every layer the book has named: raw IMU sample, Kalman update, $\widehat{\kappa}$ computation, Davis Field Equation, gain modulation, LQR feedback, actuator command. The numbers are deliberately concrete so a reader who has finished the chapters can verify each step against the formulas the book has stated. The scenario matches Lab V (Chapter 11 companion lab) reduced to a single tick.

**Setup.** The X4 reference platform: $m = 0.8$ kg, arm length $\ell = 0.12$ m, hover thrust $T_{\text{hov}} = mg \approx 7.85$ N. Control rate 1 kHz, so $\Delta t = 10^{-3}$ s. State linearized about hover: $\mathbf{x} = (x, y, z, \dot{x}, \dot{y}, \dot{z})^{\mathrm{T}} \in \mathbb{R}^6$, control $\mathbf{u} = (u_x, u_y, u_z) \in \mathbb{R}^3$ (commanded accelerations in the navigation frame, before the body-frame thrust allocation of §5.6 turns them into rotor commands).

**Step 1: raw IMU sample at $t = 10.000$ s.** The accelerometer reports specific force $\mathbf{f}_{\text{spec}} = (0.02, -0.01, 9.83)^{\mathrm{T}}$ m/s$^2$ in the body frame; the gyro reports $\boldsymbol{\omega} = (0.001, 0.001, 0.000)^{\mathrm{T}}$ rad/s. The vehicle is near hover, near level, with tiny biases. GNSS has been silent for 200 ms (the simulated drop-out from Lab V scenario B). Per the strapdown integration (§14.3), the navigation- frame specific force is $\mathbf{R}(q)\,\mathbf{f}_{\text{spec}} + \mathbf{g}_{\text{nav}}$, which at near-level attitude evaluates to approximately $(0.02, -0.01, 0.0)^{\mathrm{T}}$ m/s$^2$ of inertial

acceleration.

**Step 2: Kalman prediction.** The state covariance has been growing during the GNSS outage. Before this tick:

$$\mathbf{P}^{-} = \mathrm{diag}(0.34,\ 0.31,\ 0.05,\ \ 0.08,\ 0.07,\ 0.02)\ \ (\mathrm{m}^2 \text{ or } \mathrm{m}^2/\mathrm{s}^2).$$

After the prediction step $\mathbf{P} = \mathbf{F}\mathbf{P}^{-}\mathbf{F}^{\mathrm{T}} + \mathbf{Q}\,\Delta t$, with $\mathbf{F} = \mathbf{I}_6 + \mathbf{A}\Delta t$ for the simple double-integrator $\mathbf{A}$ and $\mathbf{Q} = \mathrm{diag}(0, 0, 0, 10^{-4}, 10^{-4}, 10^{-4})$:

$$\mathbf{P} = \mathrm{diag}(0.34016,\ 0.31014,\ 0.05004,\ \ 0.08001,\ 0.07001,\ 0.02001).$$

The trace is $\mathrm{tr}(\mathbf{P}) = 0.87035$. The baseline trace (set at startup as the nominal hover covariance, with both GNSS and IMU active) is $\mathrm{tr}(\mathbf{P}_{\mathrm{baseline}}) = 0.05$. Notice the prediction step has barely changed the trace at this tick—most of the growth has already happened across the 200 ms of accumulated drop-out.

**Step 3: $\widehat{\kappa}$ from the covariance.** Apply the design-choice formula of §10.4 with the non-negativity clamp:

$$\begin{aligned}\widehat{\kappa}(\mathbf{P}) &= \max\!\left(0,\ \frac{\mathrm{tr}(\mathbf{P})}{\mathrm{tr}(\mathbf{P}_{\mathrm{baseline}})} - 1\right) \\ &= \max\!\left(0,\ \frac{0.87035}{0.05} - 1\right) = \max(0,\ 16.41) = 16.41.\end{aligned}$$

The vehicle is 16.4 times above baseline trace; the obstruction is substantial.

**Step 4: Davis Field Equation.**

$$C = \frac{1}{1 + \widehat{\kappa}} = \frac{1}{1 + 16.41} = 0.0575.$$

Operational confidence has dropped from 1.0 (nominal hover) to about 0.058. The framework reads this as: the controller's gain modulation should be aggressive in the throttle-down direction.

**Step 5: gain modulation.** With $m_{\min} = 0.30$, $m_{\max} = 1.0$, and $\gamma = 1$ (the Lab V default):

$$m(C) \;=\; m_{\min} + (m_{\max} - m_{\min}) \cdot C^{\gamma} \;=\; 0.30 + 0.70 \cdot 0.0575 \;=\; 0.340.$$

The controller's effective gain is throttled to 34% of nominal. This is the operational consequence of the GNSS drop-out: the controller voluntarily gives up bandwidth to give the (now-degraded) estimator time to catch up.

**Step 6: LQR feedback at the throttled gain.** The reference trajectory at $t = 10$ s is $\mathbf{x}_{\text{ref}} = (0, 0, -1.5, 0, 0, 0)^{\mathrm{T}}$ (hovering at 1.5 m altitude, $z$-down convention). The estimated state is $\boldsymbol{\mu} = (0.05, -0.03, -1.48, 0.08, -0.05, 0.02)^{\mathrm{T}}$. The estimated tracking error is

$$\mathbf{e} = \boldsymbol{\mu} - \mathbf{x}_{\text{ref}} = (0.05,\; -0.03,\; 0.02,\;\; 0.08,\; -0.05,\; 0.02)^{\mathrm{T}}.$$

With the offline-stored LQR gain

$$\mathbf{K}_{\text{lqr}} = \begin{pmatrix} 2.0 & 0 & 0 & 1.8 & 0 & 0 \\ 0 & 2.0 & 0 & 0 & 1.8 & 0 \\ 0 & 0 & 5.0 & 0 & 0 & 3.0 \end{pmatrix}$$

(diagonal-dominant for a decoupled double integrator), the nominal LQR command would be

$$\mathbf{u}_{\text{nom}} = -\mathbf{K}_{\text{lqr}}\,\mathbf{e} = -(0.244,\; -0.150,\; 0.160)^{\mathrm{T}} \;\; \mathrm{m/s^2}.$$

The modulated command actually sent is

$$\begin{aligned} \mathbf{u} \;&=\; m(C) \cdot \mathbf{u}_{\text{nom}} \;=\; 0.340 \cdot (-0.244,\; 0.150,\; -0.160)^{\mathrm{T}} \\ &=\; (-0.083,\; 0.051,\; -0.054)^{\mathrm{T}} \;\; \mathrm{m/s^2}. \end{aligned}$$

The throttle has reduced the commanded corrective acceleration by a factor of about three.

**Step 7: thrust allocation.** The navigation-frame acceleration command $\mathbf{u}$ becomes a thrust magnitude $T = m|\mathbf{u} - \mathbf{g}_{\text{nav}}|$ and an attitude demand on the body-$z$ axis (§5.5). The thrust allocation matrix then turns $(T, \tau_x, \tau_y, \tau_z)$ into four rotor speeds. For this tick the dominant component is the hover thrust correction:

$$T \approx mg + m\,u_z \cdot (\text{sign}) \approx 7.85 + 0.8 \cdot (-0.054) \approx 7.81 \ \text{N}.$$

The attitude demand is tiny because the lateral commands are tiny after modulation; the rotor speeds change by less than 1% from their nominal hover values. The actuator command goes out, the physical world responds, and the loop closes for the next tick.

**Step 8: supervisor.** The Lyapunov supervisor of §20.3 evaluates $V(\mathbf{e}) = \mathbf{e}^{\mathrm{T}}\mathbf{P}_{\text{inf}}\mathbf{e}$ on the estimated tracking error, augmented by the covariance trace allowance:

$$V_{\text{cert}} = V(\mathbf{e}) + \text{tr}(\mathbf{P}) \approx 0.21 + 0.87 = 1.08.$$

The envelope threshold is $V_{\text{threshold}} = 8.5$ (set from the LQR cost evaluated at the envelope boundary $\mathbf{e}_{\text{env}} = (1, 1, 0.5, 0.5, 0.5, 0.5)$). The supervisor stays quiet: the certified envelope $V_{\text{cert}} = 1.08 \ll 8.5$.

**What this cycle has shown.** The framework's pipeline ran end to end in well under 1 ms:

- Raw IMU → Kalman prediction: $\mathbf{P} \in \mathbb{R}^{6\times 6}$ updated.
- Kalman covariance → scalar obstruction: $\widehat{\kappa} = 16.41$.
- Davis Field Equation: $C = 0.058$.
- Gain modulation: $m(C) = 0.340$.
- Modulated LQR command:
  $\mathbf{u} = (-0.083, 0.051, -0.054)^{\mathrm{T}}$ m/s$^2$.

- Thrust allocation: $T = 7.81$ N plus a small attitude torque.
- Supervisor envelope check: quiet.

Every chapter of this book has been about one of those bullet points or about the geometric reading that made it the right computation to perform. The throttle ratio $m(C) = 0.340$ is what holonomy-gated control *is*—one number, computed from one scalar $\widehat{\kappa}$, applied to one nominal gain $\mathbf{K}_{\mathrm{lqr}}$, and the framework's central operational claim is that this single multiplication is sufficient for the gain-modulation decision the controller has to make on this tick (§17.3 for the scope of the sufficiency claim).

Chapter G

# *References*

The book cites works inline throughout the chapters. This chapter collects the full bibliographic entries, organised alphabetically by first author. Entries marked with ⋆ are the framework's own foundational papers; the rest are external references the book draws on.


**Agrachev, A. A. and Sachkov, Y. L.** (2004). *Control Theory from the Geometric Viewpoint.* Encyclopaedia of Mathematical Sciences 87. Springer, Berlin. (Appendix C, Appendix E.)

**Amari, S.** (2016). *Information Geometry and Its Applications.* Applied Mathematical Sciences 194. Springer, Tokyo. (Chapter 10, Appendix D.)

**Amari, S. and Nagaoka, H.** (2000). *Methods of Information Geometry.* Translations of Mathematical Monographs 191. American Mathematical Society, Providence, RI. (Appendix D.)

**Ambrose, W. and Singer, I. M.** (1953). "A Theorem on Holonomy." *Transactions of the American Mathematical Society* 75(3): 428–443. (Chapter 3, Marcella chapter.)

**Bernoulli, J.** (1696). "Problema novum ad cuius solutionem Mathematici invitantur" (posing the brachistochrone problem). *Acta Eruditorum*, June 1696. (Chapter 23 epigraph.)

**Bode, H. W.** (1945). *Network Analysis and Feedback Amplifier Design.* D. Van Nostrand, New York. (Chapter 7.)

**Bryson, A. E. and Ho, Y.-C.** (1969). *Applied Optimal Control: Optimization, Estimation, and Control.* Blaisdell, Waltham, MA. (Chapter 8, Chapter 23, Appendix E.)

**Bullo, F. and Lewis, A. D.** (2005). *Geometric Control of Mechanical Systems.* Texts in Applied Mathematics 49. Springer, New York. (Appendix E.)

⋆ **Davis, B. R.** (2025). "A Davis-Geometric Reading of the Yang–Mills Mass Gap." Zenodo DOI: 10.5281/zenodo.17846521. (Chapter 2, Chapter 25.)

⋆ **Davis, B. R.** (April 2026). "The Dual-Bundle Theorem and the Holonomy Sandwich." Standalone manuscript in preparation. (Marcella chapter §21.5.)

⋆ **Davis, B. R.** (2026). "The Davis Duality of Approximation and Obstruction." Zenodo DOI: 10.5281/zenodo.19428406. (Used throughout; introduced in Chapter 1.)

⋆ **Davis, B. R.** (2026). "The Bra Strap Principle: Local-to-Global Propagation in Fiber Bundles." Zenodo DOI: 10.5281/zenodo.20273573. (Chapter 1, Chapter 3, Chapter 16, Chapter 18.)

**Doyle, J. C.** (1978). "Guaranteed Margins for LQG Regulators." *IEEE Transactions on Automatic Control* AC-23(4): 756–757. (Chapter 9 epigraph; Chapter 9 §9.0.)

**Doyle, J. C., Francis, B. A. and Tannenbaum, A. R.** (1992). *Feedback Control Theory.* Macmillan, New York. (Reprinted by Dover, 2009.) (Chapter 7.)

**Doyle, J. C. et al.** (1989). "State-Space Solutions to Standard

$H_2$ and $H_\infty$ Control Problems," with K. Glover, P. P. Khargonekar, and B. A. Francis. *IEEE Transactions on Automatic Control* 34(8): 831–847. (Chapter 9.)

**Harpold, J. C. and Graves, C. A.** (1979). "Shuttle Entry Guidance." *Journal of the Astronautical Sciences* 27(3): 239–268. (Chapter 23.)

**Ioannou, P. A. and Kokotović, P. V.** (1983). *Adaptive Systems with Reduced Models.* Lecture Notes in Control and Information Sciences 47. Springer, Berlin. (Chapter 13; §13.2.)

**Kalman, R. E.** (1960a). "Contributions to the Theory of Optimal Control." *Boletín de la Sociedad Matemática Mexicana* 5(2): 102–119. (Chapter 8.)

**Kalman, R. E.** (1960b). "A New Approach to Linear Filtering and Prediction Problems." *Journal of Basic Engineering* (Transactions of the ASME, Series D) 82(1): 35–45. (Chapter 10.)

**Kalman, R. E. and Bucy, R. S.** (1961). "New Results in Linear Filtering and Prediction Theory." *Journal of Basic Engineering* 83(1): 95–108. (Chapter 10.)

**Khalil, H. K.** (2002). *Nonlinear Systems.* 3rd edition. Prentice Hall, Upper Saddle River, NJ. (Chapter 11 epigraph.)

**Kleinman, D. L.** (1968). "On an Iterative Technique for Riccati Equation Computations." *IEEE Transactions on Automatic Control* 13(1): 114–115. (Chapter 8.)

**Kobayashi, S. and Nomizu, K.** (1963, 1969). *Foundations of Differential Geometry, Volumes I and II.* Wiley-Interscience, New York. (Standard reference for fiber-bundle geometry; Chapter 3.)

**Lyapunov, A. M.** (1892). "The General Problem of the Stability of Motion." Doctoral dissertation, Kharkov University. English translation: A. T. Fuller (transl.), Taylor & Francis, London, 1992. (Chapter 11.)

**Mellinger, D. and Kumar, V.** (2011). "Minimum Snap Trajectory Generation and Control for Quadrotors." *IEEE International Conference on Robotics and Automation (ICRA) 2011*, Shanghai, pp. 2520–2525. (Chapter 23.)

**Montgomery, R.** (2002). *A Tour of Sub-Riemannian Geometries, Their Geodesics and Applications.* Mathematical Surveys and Monographs 91. American Mathematical Society, Providence, RI. (Appendix C.)

**Murray, M. K. and Rice, J. W.** (1993). *Differential Geometry and Statistics.* Monographs on Statistics and Applied Probability 48. Chapman & Hall, London. (Chapter 10.)

**Parks, P. C.** (1966). "Lyapunov Redesign of Model Reference Adaptive Control Systems." *IEEE Transactions on Automatic Control* AC-11(3): 362–367. (Chapter 12.)

**Pontryagin, L. S. et al.** (1962). *The Mathematical Theory of Optimal Processes*, with V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. English translation: K. N. Trirogoff (transl.), L. W. Neustadt (ed.), Interscience, New York. (Chapter 22, Appendix E.)

**Saint-Exupéry, A. de** (1939). *Wind, Sand and Stars.* (Original French: *Terre des hommes.*) Reynal & Hitchcock, New York. (Chapter 1.)

**Skogestad, S. and Postlethwaite, I.** (2005). *Multivariable Feedback Control: Analysis and Design.* 2nd edition. Wiley, Chichester. (Chapter 7.)

**Smith, R. C., Self, M. and Cheeseman, P.** (1990). "Estimating Uncertain Spatial Relationships in Robotics." In I. J. Cox and G. T. Wilfong (eds.), *Autonomous Robot Vehicles*, pp. 167–193. Springer, New York. (Chapter 16.)

**Stein, G.** (1989). "Respect the Unstable." Bode Lecture, IEEE Conference on Decision and Control. Reprinted in *IEEE Control Systems Magazine* 23(4): 12–25, August 2003. (Chapter 7 epigraph.)

**Stevens, B. L., Lewis, F. L. and Johnson, E. N.** (2015). *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems.* 3rd edition. Wiley, Hoboken, NJ. (Chapter 5, Chapter 6.)

**Zames, G.** (1981). "Feedback and Optimal Sensitivity: Model Reference Transformations, Multiplicative Seminorms, and Approximate Inverses." *IEEE Transactions on Automatic Control* AC-26(2): 301–320. (Chapter 9.)

**Zarchan, P.** (2019). *Tactical and Strategic Missile Guidance.* 7th edition. Progress in Astronautics and Aeronautics 239. AIAA, Reston, VA. (Original first edition: AIAA 1990.) (Chapter 23.)

**Further reading and historical references named in the text.** Several historical names are mentioned in the chapters without full bibliographic citation (their work is the historical background rather than a directly used source): Atiyah (characteristic classes); Brouwer (homotopy theory and fixed-point theorems); Ehresmann (the fiber-bundle formulation in his 1950s lecture notes); Steenrod (the 1951 *Topology of Fibre Bundles*, Princeton University Press); Levi-Civita (the original parallel-transport construction, 1917); Ricci (the curvature tensor, 1900); Krasovskii, Letov, and Pontryagin (the mid-twentieth-century

Russian school of nonlinear control); Sastry, Vidyasagar, Khalil (standard nonlinear-control textbooks); Crassidis and Markley (*Fundamentals of Spacecraft Attitude Determination and Control*, Springer, 2014); Bellman (*Dynamic Programming*, Princeton, 1957); Laub (Schur-method Riccati solvers, *IEEE TAC* 24(6), 1979); Skovgaard (early information-geometric covariance Lie algebra work, 1984); Barbalat (Barbalat's lemma, foundational for MRAC stability proofs). Full citations for these are available in the standard textbooks they are most often cited through.

**Davis-Geometric framework papers.** The framework's own papers are catalogued in full at the Davis-Wilson Lattice repository and at `davisgeometric.com`. The three foundational papers this book draws on directly are marked with $\star$ above. The broader catalogue includes structural readings of Navier–Stokes regularity, P-versus-NP, Poincaré (Perelman's proof), Riemann hypothesis, and the field-equations-of-semantic-coherence work.

# *About the Author*

**Bee Rosa Davis (she/her)** is an applied mathematics researcher, software engineer, and Trinidadian-American aerospace systems builder whose work bridges differential geometry, control theory, and the engineering of safety-critical systems. She holds an MS in Digital Forensics, Counterintelligence, and Cybersecurity from Brown University, an MA in Communication from the University of the Pacific, and a BA in Logic and Analytics from Morehouse College, where she was a Martin Luther King Jr. Scholar.

She has no PhD. She has no institutional affiliation. What she has is a body of work that speaks for itself, a flight-control architecture she designed from first principles, and a dog named Teddy who showed her—one afternoon at the kitchen table while she was working on a serialization format—that a fiber bundle is just a base manifold with things attached.

*The Professional Path*

Before the mathematics in this book had a name, there was a career built on getting systems to not fail when failure would have cost something irreversible. At NASA Johnson Space Center, in the Flight Operations Directorate, she designed high-throughput telemetry pipelines processing 2.3 terabytes daily with sub-15-millisecond latency, contributed to mission control simulation work for aerospace hardware from fuses to full launch stacks, and helped architect secure data pipelines for the James Webb

Space Telescope. Those roles—where simulation, telemetry, and lives-on-the-line reliability converge—are where the framework's commitment to *recovery authority sized before the maneuver* comes from. The kid in the driveway who sized her brakes before riding became the engineer who would not ship a flight-control system without an envelope-protection layer.

At IBM X-Force Red, she served as Principal Adversarial Intelligence Engineer, building platforms for real-time threat analysis and automated vulnerability discovery, including the geometric attack-surface methods through which she discovered and disclosed four Windows kernel zero-day vulnerabilities to the Microsoft Security Response Center in late 2025. At Humana, she led quantum-resistant encryption development for healthcare data protection. Before that: Pandora Radio, where as a founding engineer she architected machine-learning systems that grew ad revenue 750%. Before that: the intelligence community—NSA, Naval Intelligence, DISA, Army Intelligence—building anomaly-detection systems for environments where getting it wrong meant someone did not come home.

The throughline was always geometry. Telemetry pipelines are manifolds. Mission-control trajectories are sections of coupled fiber bundles. Anomaly detection on encrypted channels is curvature measurement on signal spaces. Vulnerability surfaces are obstructions in a higher-dimensional code manifold. The mathematics was always there. This book is one of the volumes where it gets named.

*The Research Program*

Her published research develops rigorous mathematical frameworks for how systems—physical, computational, biological—organize information and fail. The Davis-Geometric series this book belongs to includes: the *Bra Strap Principle* (a propagation

theorem for sections of fiber bundles over $\mathbb{S}^1$, Zenodo 2026); the *Davis Duality of Approximation and Obstruction* (a two-sided bound on flat-tangent error in curved bundles, Zenodo 2026); the *Davis Manifold* (Riemannian state spaces with compositional error budgets, Zenodo 2025); and an information-geometric reduction of the Yang–Mills mass gap (Zenodo 2025, with a $15\sigma$ empirical confirmation on a discretized $8^4$ lattice). The framework also reads Navier–Stokes regularity, P-versus-NP separation, Perelman's Ricci flow, and Riemann's non-trivial zeros as instances of the same Davis Field Equation $C = \tau/K = 1/(1+\hat{\kappa})$, with each reading reported as a separate consistency check rather than a claim of formal resolution.

The engineering thesis of *this* book is ICARUS—the Integrated Curvature-Aware Routing and Uncertainty Substrate—filed as U.S. Provisional Patent Application 64/045,891 in April 2026, alongside its supporting database engine GIGI (U.S. Provisional 64/045,889) and serialization layer DHOOM (U.S. Provisional 64/008,940, filed March 2026). ICARUS is committed at the schema layer to defensive applications only: the architecture's type system does not admit the actuator vectors required for weaponization. That choice, like every other choice in the book, is the same kid's choice in the driveway, formalized.

*The Books*

*The Geometry of Flight* is the fifth volume in her ongoing series. The earlier volumes are *Hidden Variable: Unlocking Patterns in a World Obsessed with Structure* (2025; about her mother, the diagnosis-of-exclusion that structured her early life, and the pattern-recognition methodology she built in response); *The Geometry of Sameness: Riemannian Equivalence of Translation and Distance for Semantic Detection* (December 2025; Book 2 of the *Geometric Intelligence* sub-series, on which the present book's globes-and-maps reading depends); *The Geometry of Fuel*

(on energy as a configuration manifold); and *Superfluid* (on phase transitions and criticality, drawing the architecture from Landau's 1962 Nobel work). She also writes science fiction (the Astralis series: *Fly Into the Sun*, *A Pale Jewel*) and memoir—*Motion to Be Seen: One Black Trans Woman's Fight for Algorithmic Justice* reached #1 on Amazon as a New Release. Forthcoming volumes include a standalone MARCELLA manuscript (the transformer-replacement architecture Chapter 21 introduces) and the formal write-ups of each cross-domain consistency check the Davis Field Equation has been used to read.

*The Person*

She is a Black trans woman, a lesbian, and a proud Trinidadian-American who wants her country—and its technologies—to live up to their promises. She is the Overall Mother of the Kiki House of Diwa, where she guides the next generation with the same care she brings to her systems work—watching the body, reading the geometry, knowing when someone is about to fall before they fall. She raised three children she had in her early twenties, who arrived exactly when the world might have expected wild, drifting years and instead taught her to grow up early, ask better questions, and build systems that could hold weight. She rebuilt PyTorch from scratch (DavisTensors) to understand what she was using. She taught herself differential geometry, spectral theory, category theory, and gauge theory from textbooks and papers because the questions would not leave her alone.

She publishes research under the handle `@nurdymuny` and keeps building systems—and stories—where people, and machines, can be safely and stubbornly themselves across time, context, and representation. She lives in Sacramento with Teddy.

The mathematics in this book was developed without institutional support, without collaborators, without the safety net that

credentials provide. It was developed because the patterns were there, visible to anyone willing to look carefully enough, and someone needed to write them down. If this book helps you see your own systems, your own aircraft, or your own life as a little more navigable—and a little more honest about what it means to size your brakes before the engine starts—then it has done what its author hoped it would do.

*The geometry does not ask who you are.*
*It asks only whether you were paying attention.*