Pedagogical Framework and Curriculum Design for Robotic Navigation Backends: Filter-Based and Optimization-Based State Estimation

Designing a comprehensive curriculum in autonomous mobile robotics requires balancing historical algorithmic foundations with contemporary state-of-the-art implementations[1]. The navigation backend—the computational engine responsible for state estimation, localization, and mapping—has evolved through two major historical paradigms: recursive filter-based architectures and non-linear optimization-based architectures[3]. Educating students in these backends demands a structured progression that couples mathematical rigor with hands-on implementation[1]. Filter-based approaches introduce foundational concepts of probability density functions, Markovian state transitions, and Gaussian error propagation[1]. Conversely, optimization-based approaches expose students to sparse matrix factorization, Lie algebra, maximum a posteriori (MAP) inference, and graphical models[1].

This resource guide synthesizes pedagogical strategies, theoretical developments, lecture sequences, primary text recommendations, and laboratory implementations for teaching filter-based and optimization-based backends in graduate and upper-level undergraduate robotics courses[1].

Theoretical Evolution and Pedagogical Mechanics

Understanding the transition from recursive Bayesian filters to global factor graph optimization requires examining how both approaches handle non-linearity, computational memory, and statistical error over time[4]. Methodologically, state estimation has shifted from sequential, single-point linearized filters—where historical sensor data is discarded after covariance updates—to batch and sliding-window factor graphs that retain trajectory variables to permit continuous re-linearization[3].

Recursive filtering methods, such as the Extended Kalman Filter (EKF), enforce strict temporal Markov assumptions[3]. The system evaluates motion and measurement models sequentially, maintaining only the current state estimate $\hat{\mathbf{x}}_k$ and its associated error covariance matrix $\mathbf{P}_k$ [3]. Non-linearities in spatial transformations are handled via first-order Taylor expansions around the prior state estimate[3]:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k, \quad \mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k, \quad \mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$$

$$\mathbf{F}_k = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1,k-1}}, \quad \mathbf{H}_k = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k,k-1}}$$

Because linearization occurs once per time step around the propagated state estimate, early linearization errors cannot be corrected[4]. If the initial state uncertainty is large or sensor noise deviates from Gaussian distributions, the estimated covariance $\mathbf{P}_k$ becomes overconfident, leading to inconsistency and divergence[4].

Optimization-based backends mitigate this failure mode by reformulating state estimation as a Maximum A Posteriori (MAP) problem over a trajectory or sliding window[5]:

$$\mathbf{X}^* = \arg\max_{\mathbf{X}} \prod_i f_i(\mathbf{X}_i) = \arg\min_{\mathbf{X}} \sum_i \frac{1}{2} \| h_i(\mathbf{X}_i) - \mathbf{z}_i \|^2_{\mathbf{\Sigma}_i}$$

Because the optimization engine retains historical states within a factor graph, it re-linearizes non-linear measurement functions $h_i(\mathbf{X}_i)$ around updated estimates during every solver iteration, such as Gauss-Newton or Levenberg-Marquardt routines[5]. This capability preserves numerical stability and prevents premature convergence in non-convex spatial domains[5].

The computational bottleneck of classic feature-based EKF-SLAM stems from the dense coupling of state variables[3]. In a two-dimensional environment with N landmark locations and a three-degree-of-freedom robot pose, the joint state vector dimension is $(3 + 2N)$ [3]. Updating the full covariance matrix $\mathbf{P}_k$ requires updating every cross-correlation term between the robot pose and all map features[3]. This imposes a computational complexity of $\mathcal{O}(N^2)$ per time step and a spatial memory footprint of $\mathcal{O}(N^2)$, making pure EKF-SLAM computationally prohibitive for large-scale maps[3].

Factor graph optimization transforms this dense problem into a sparse graph inference task[5].

State variables form variable nodes, while spatial constraints form factor nodes[6]. The corresponding measurement matrix displays structural sparsity because individual sensor observations constrain only small subsets of variables[6]. By framing state estimation as a sparse matrix linear algebra problem ($\mathbf{A}^T\mathbf{A}\Delta\mathbf{x} = -\mathbf{A}^T\mathbf{b}$), solvers exploit variable elimination techniques, Cholesky factorization, and QR decomposition[5]. Modern algorithms such as incremental Smoothing and Mapping (iSAM2) update only the affected nodes in a data structure known as a Bayes Tree, reducing execution time to an amortized cost of $\mathcal{O}(1)$ for most local tracking steps[5].

Filter-Based Navigation Resources and Instructional Modules

Teaching filter-based estimation establishes essential student competency in state representation, probability theory, kinematic modeling, and noise characterization[1]. Primary foundational textbooks provide the mathematical framing necessary for understanding recursive state estimation[1]. The canonical text *Probabilistic Robotics* by Sebastian Thrun, Wolfram Burgard, and Dieter Fox devotes Chapter 3 to non-linear Bayes filtering and Chapter 10 to the detailed mathematical derivation, data association routines, and structural limitations of EKF-SLAM[4]. In *State Estimation for Robotics*, Timothy D. Barfoot provides rigorous coverage of Gaussian probability theory, linear-Gaussian estimation, and non-linear filtering variations, including Extended Kalman Filtering and Sigmapoint or Unscented Kalman Filtering across Chapters 2 through 4[1]. Additionally, *Autonomous Mobile Robots* by Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza offers targeted coverage of wheel kinematics, dead reckoning drift, and EKF localization against known landmark maps in Chapters 4 and 5[12]. Open-access lecture materials enhance these textual references by demonstrating step-by-step mathematical derivations[7]. Dr. Cyrill Stachniss's *Robot Mapping* course provides open video lectures and slide decks covering Bayes filters, linear Kalman filters, Extended Kalman filters, and feature-based EKF-SLAM, detailing state space expansions, Jacobian derivations, and landmark initialization routines[3]. Complementary open courseware from ETH Zurich's Autonomous Systems Lab delivers slide packages focusing on mobile robot kinematics, odometry error propagation, and practical sensor calibration for Kalman filtering applications[12]. To internalize recursive filter mechanics, students must construct implementations from scratch in matrix computation environments like Python, MATLAB, or Octave[7]. Standard laboratory assignments include two-dimensional EKF-SLAM implementations derived from Cyrill Stachniss's repository sheet solutions, where students write motion model updates, measurement Jacobians, and dynamic covariance expansions[3]. Further hands-on exercises based on Tim Barfoot's curriculum guide students through process noise tuning, non-linear velocity motion models, and Unscented Transform sigmapoint generation using libraries such as FilterPy[7].

Unit / Module	Learning Objectives	Key Theoretical Concepts	Reference Literature & Lecture Media	Recommended Practical Assignment
1. Recursive Bayes Filtering	Formulate state estimation probabilistically under Markov assumptions[3].	Prediction/Correction loops, Chapman-Kolmogorov equations, Gaussian distributions[3].	Thrun Ch. 2–3[13]; Stachniss Lec. 03[19].	Implement a 1D discrete Bayes filter for mobile robot localization.
2. Extended Kalman Filter (EKF)	Handle continuous non-linear kinematic and sensor models[3].	First-order Taylor series, Jacobians, error covariance propagation[1].	Barfoot Ch. 4[1]; Stachniss Lec. 04[19].	Build a 2D robot pose tracker using wheel odometry and range beacons[3].
3. Feature-Based EKF-SLAM	Estimate robot pose and landmark positions concurrently[3].	State vector expansion, spatial cross-correlation matrices, data association[3].	Thrun Ch. 10[4]; Stachniss Lec. 05[3].	Implement a full 2D range-bearing EKF-SLAM in Python or Octave[3].
4. Unscented Kalman Filter (UKF)	Avoid explicit Jacobian calculations in highly non-linear dynamics[1].	Sigmapoint sampling, Unscented Transform, mean/covariance reconstruction[1].	Barfoot Ch. 4[1]; Stachniss Lec. 06[7].	Implement UKF for non-linear vehicle chassis tracking; compare drift against EKF[7].

Optimization-Based Navigation Resources and Instructional Modules

Optimization-based approaches represent the core architecture for contemporary robotics systems, including Visual-Inertial Odometry (VIO), LiDAR-Inertial Odometry (LIO), and

large-scale graph SLAM[5]. *Factor Graphs for Robot Perception* by Frank Dellaert and Michael Kaess serves as the foundational text for graphical state estimation models[5]. The monograph provides comprehensive coverage of factor graph construction, variable elimination, Bayes Trees, incremental smoothing via iSAM and iSAM2, and multi-sensor fusion abstractions[5]. Additionally, Chapters 5, 8, and 9 of Timothy D. Barfoot's *State Estimation for Robotics* address batch non-linear continuous-time trajectory estimation, pose-graph relaxation, Lie groups ($SE(2)$ and $SE(3)$), and sparse matrix factorization[1].

Lectures on graph-based backends focus on formulating optimization problems over spatial network representations[5]. Frank Dellaert's *Robot Perception* courseware at the Georgia Institute of Technology presents factor graph theory, GTSAM library usage, and visual mapping backends[28]. Cyrill Stachniss's lecture modules on graph-based SLAM cover non-linear least-squares error minimization, information matrices, pose-graph construction, and loop closure constraint optimization[7].

Practical implementation requires training students on industry-standard factor graph optimization libraries[6]. The Georgia Tech Smoothing and Mapping (GTSAM) BSD-licensed C++ library, equipped with Python bindings, provides pre-built factor nodes for odometry, range-bearing landmarks, IMU pre-integration, and visual projections[6]. Tutorial repositories such as gtsam_tutorials re-implement canonical graph optimization problems in Python, enabling students to construct Pose Graph SLAM and landmark-based factor networks[8]. Advanced academic assignments, such as those from MIT's Visual Navigation for Autonomous Vehicles (VNAV) course, challenge students to build fixed-lag smoothers and visual-inertial fusion pipelines using GTSAM abstractions[25].

Unit / Module	Learning Objectives	Key Theoretical Concepts	Reference Literature & Lecture Media	Recommended Practical Assignment
1. Non-Linear Least Squares (NLS)	Formulate spatial optimization as unconstrained objective minimization[5].	Residual vectors, information matrices, Gauss-Newton, Levenberg-Marquardt[5].	Barfoot Ch. 5[1]; Stachniss Lec. 10[7].	Implement a standard Gauss-Newton solver for 2D point cloud registration[7].
2. Factor Graph Fundamentals	Convert joint probability distributions into bipartite	Variable nodes, factor nodes, MAP estimation,	Dellaert & Kaess Ch. 1–3[5]; GTSAM Docs[25].	Construct a GTSAM factor graph for simulated 2D

	graphical models[5].	canonical Gaussian forms[6].		planar robot trajectories[8].
3. Pose Graph SLAM & Loop Closures	Correct trajectory drift using spatial relative transformation constraints[5].	Lie groups ($SE(2)/SE$), loop closure factors, robust loss functions (Huber, Cauchy)[1].	Dellaert & Kaess Ch. 4[5]; Stachniss Lec. 11[7].	Implement Python Pose-Graph SLAM using GTSAM to resolve drift on benchmark datasets[8].
4. Incremental Smoothing & Mapping	Enable real-time sliding window optimization and continuous updates[5].	Variable elimination, Bayes Tree, iSAM2, marginalization priors[5].	Dellaert & Kaess Ch. 5–6[5]; MIT VNAV Lab 7[30].	Build a real-time sliding-window VIO/LIO backend factor graph using iSAM2[6].

Comparative Engineering Analysis for Advanced Instruction

When training graduate students or industry practitioners, instructors must highlight the clear design trade-offs between filter-based and optimization-based architectures[4]. The choice of navigation backend depends directly on available compute resources, sensor modalities, system dynamic requirements, and operational domain duration[6]. Filter-based methods provide low-memory, fixed computational bounds suited for real-time tracking on embedded microcontrollers, but they suffer from unrecoverable linearization drift[4]. Optimization-based methods incur higher memory costs to store historical state factors, but offer continuous re-linearization, multi-modal robust loss functions, and global trajectory consistency[5].

Technical Metric / Feature	Filter-Based Backends (EKF / UKF)	Optimization-Based Backends (Factor Graphs / iSAM2)
Mathematical Formulation	Recursive Bayesian filtering; sequential conditional probability updates[1].	Maximum A Posteriori (MAP) estimation over variable networks[5].

Linearization Strategy	Single-point linearization around current prior estimate; unrecoverable error accumulation[4].	Iterative re-linearization of all graph variables around updated optimization estimates[5].
Computational Complexity	$\mathcal{O}(N^2)$ or $\mathcal{O}(N^3)$ per step for N map landmarks; $\mathcal{O}(1)$ for state-only filtering[3].	$\mathcal{O}(1)$ amortized local updates via Bayes Tree sparsity; $\mathcal{O}(V^3)$ worst-case full batch[5].
Memory Consumption	Low and constant for fixed state vectors; grows quadratically $\mathcal{O}(N^2)$ in landmark SLAM[3].	Linear in historical variables and constraints $\mathcal{O}(V +$; requires marginalization routines[6].
Multi-Sensor Fusion Capability	Requires tight temporal alignment; tightly coupled via process/measurement Jacobians[3].	Modular; arbitrary asynchronous measurements added directly as factor nodes[6].
Outlier Handling & Robustness	Sensitive to false data association; requires rigid Chi-squared gating[1].	Highly robust when using non-linear loss functions (Huber, DCS, Max-Mixture)[5].
Primary Domain Application	High-rate embedded state estimation, flight controllers, short-term visual odometry[11].	Large-scale mapping, multi-session SLAM, Visual-Inertial/LiDAR Odometry backends[6].

Integrated Laboratory Curriculum and Assessment Design

To provide students with a cohesive learning experience, instructors can structure a semester curriculum around a progressive, unified codebase that transitions systematically from kinematic tracking to sliding-window graph optimization[2].

The initial block of the practical curriculum, spanning Weeks 1 through 6, establishes the fundamentals of recursive filtering[3]. During Weeks 1 and 2, students implement differential-drive and omnidirectional kinematic models to observe dead reckoning drift[12]. In Weeks 3 and 4, students construct an Extended Kalman Filter that fuses wheel odometry with

sparse landmark updates[3]. Weeks 5 and 6 expand this system into full two-dimensional EKF-SLAM[3]. By implementing dynamic covariance expansions, students directly observe execution slowing and covariance divergence caused by linearization errors during sharp maneuvers[3].

The intermediate block, covering Weeks 7 through 12, transitions the class into non-linear least squares and graphical optimization[5]. Weeks 7 and 8 focus on building Gauss-Newton and Levenberg-Marquardt solvers from scratch to perform point cloud registration[5]. Weeks 9 and 10 introduce factor graph abstractions using GTSAM, allowing students to rebuild their two-dimensional trajectory estimators and compare computational scaling directly against their previous EKF code[3]. Weeks 11 and 12 expand these graphs into Pose-Graph SLAM using real-world benchmark datasets such as KITTI or Victoria Park, incorporating spatial loop closures and robust loss functions to eliminate false constraints[5].

The final block, spanning Weeks 13 and 14, culminates in an advanced sensor fusion capstone project[6]. Students construct a real-time fixed-lag smoother using GTSAM or iSAM2, fusing high-rate IMU pre-integration factors with visual keyframe features or LiDAR odometry estimates[6]. Assessment evaluates the student's ability to marginalize historical state variables accurately while preserving prior covariance estimates[5].

Strategic Guidance for Course Delivery

Teaching state estimation backends effectively requires bridging abstract probabilistic mathematics with practical software engineering[1]. Instructors should standardize on Python for introductory filtering assignments to minimize software setup overhead[8]. When introducing factor graphs, leveraging GTSAM's Python bindings (gtsam) offers high performance while eliminating low-level compilation barriers[6].

Students gain deeper technical insight when algorithms fail under controlled conditions[4]. Course assignments should deliberately introduce high non-linearities, unobservable system directions, inaccurate initializations, and non-Gaussian measurement noise[4]. Experiencing an EKF covariance explosion or observing a factor graph solver trap in a local minimum teaches crucial diagnostic and debugging skills[4].

Finally, instruction should contextualize filtering not as an obsolete methodology, but as an efficient state estimation tool optimized for resource-constrained embedded microcontrollers[11]. Conversely, factor graph optimization should be presented as the standard architecture powering modern spatial computing, autonomous vehicles, and multi-sensor fusion pipelines[6].

Works cited

1. State Estimation for Robotics - Timothy D. Barfoot - Google Books, https://books.google.it/books?id=EpsqDwAAQBAJ
2. UvA-DARE (Digital Academic Repository), https://pure.uva.nl/ws/files/298294653/DesigningVisionForAutonomousRobots.pdf
3. CSE-571 Robotics,

https://courses.cs.washington.edu/courses/cse571/22sp/slides/10-mapping-slam.pdf
4. EKF SLAM, http://ais.informatik.uni-freiburg.de/teaching/ws15/mapping/pdf/slam05-ekf-slam.pdf
5. [PDF] Factor Graphs for Robot Perception | Semantic Scholar, https://www.semanticscholar.org/paper/Factor-Graphs-for-Robot-Perception-Dellaert-Kaess/d30c40f372976914f6a433d9bd6c70a2eea54832
6. Factor Graph Optimization vs EKF: Robotics State Estimation, https://industrialmonitordirect.com/blogs/knowledgebase/factor-graph-optimization-vs-ekf-state-estimation-reference
7. conorhennessy/SLAM-Course-Solutions - GitHub, https://github.com/conorhennessy/SLAM-Course-Solutions
8. AnshShah3009/gtsam_tutorials - GitHub, https://github.com/AnshShah3009/gtsam_tutorials
9. SLAM using Extended Kalman Filter on a Robot for Localization and, https://www.slideshare.net/slideshow/slam-using-extended-kalman-filter-on-a-robot-for-localization-and-mappings/270137265
10. Factor Graph Accelerator for LiDAR-Inertial Odometry (Invited Paper), https://horizon-lab.org/pubs/iccad22.pdf
11. EKF SLAM – Simultaneous Localization and Mapping, https://www.ipb.uni-bonn.de/html/teaching/photo12-2021/2021-pho2-16-ekf-slam.pptx.pdf
12. (PDF) Intro to Autonomous Mobile Robots - Academia.edu, https://www.academia.edu/3491049/Intro_to_Autonomous_Mobile_Robots
13. Taeyoung96/SLAM-Resources-for-Beginner - GitHub, https://github.com/Taeyoung96/SLAM-Resources-for-Beginner
14. STATE ESTIMATION FOR ROBOTICS, https://mingkangxiong.github.io/assets/books/State_Estimation_for_Robotic_2018.pdf
15. STATE ESTIMATION FOR ROBOTICS - University of Toronto, https://asrl.utias.utoronto.ca/~tdb/bib/barfoot_ser17.pdf
16. Introduction and Lecture Overview Autonomous Mobile Robots, https://www.cs.columbia.edu/~allen/F19/NOTES/Lecture_2.pdf
17. 9 - Localization - Kalman Filter | PDF | Kalman Filter | Mathematics, https://www.scribd.com/document/1019208175/9-Localization-Kalman-Filter
18. Autonomous Mobile Robots Course Overview - Kalman Filter - Scribd, https://www.scribd.com/presentation/909771196/1-Introduction-Slides
19. SLAM Course - 03 - Kalman Filter - Cyrill Stachniss - YouTube, https://www.youtube.com/watch?v=ELyQMGF_Wlg
20. SLAM-Course - 04 - Extended Kalman Filter (2013/14 - YouTube, https://www.youtube.com/watch?v=DE6Jn2cB4J4
21. SLAM Course - 04 - EKF SLAM - Cyrill Stachniss - YouTube, https://www.youtube.com/watch?v=AdjtKlg_bWw
22. Cyrill Stachniss - EKF SLAM (2013/14 - YouTube,

https://www.youtube.com/watch?v=XeWG5D71gC0
23. Syllabus | State Estimation for Robotics and Autonomous Systems, https://birdjj.github.io/courses-estimation/syllabus/
24. The exercises in Barfoot's book: state estimation for robotics - GitHub, https://github.com/gaoxiang12/state-estimation-exercises
25. Tutorials — GTSAM 4.0.2 documentation - Read the Docs, https://gtsam-jlblanco-docs.readthedocs.io/en/latest/Tutorials.html
26. State Estimation for Robotics - eBooks.com, https://www.ebooks.com/en-gb/book/95787653/state-estimation-for-robotics/timothy-d-barfoot/
27. [PDF] State Estimation for Robotics | Semantic Scholar, https://www.semanticscholar.org/paper/State-Estimation-for-Robotics-Barfoot/7d29c8cee358616004bdfbdbce6019d49abd6f2c
28. Good references to get started with factor graph optimisation - Reddit, https://www.reddit.com/r/ControlTheory/comments/1lzs450/good_references_to_get_started_with_factor_graph/
29. Tutorial | GTSAM, https://gtsam.org/tutorials/
30. Exercises | VNAV, https://vnav.mit.edu/labs/lab7/exercises.html
31. A Factor-Graph Approach for Optimization Problems with Dynamics, https://mandyxie.github.io/dfg_dfgp.pdf
32. Factor Graphs for Robot Perception - ResearchGate, https://www.researchgate.net/publication/319132170_Factor_Graphs_for_Robot_Perception
33. Visual SLAM for Autonomous Navigation of MAVs, https://hsbiblio.uni-tuebingen.de/xmlui/bitstream/handle/10900/55094/Dissertation_Yangs.pdf?sequence=1&isAllowed=y
34. VIZARD: Reliable Visual Localization for Autonomous Vehicles in, https://arxiv.org/html/1902.04343v2
35. [GTSAM python Tutorial] Robust Pose-graph Optimization - YouTube, https://www.youtube.com/watch?v=zOr9HreMthY
36. GTSAM 4.0 Tutorial Theory, Programming, and Applications, https://dongjing3309.github.io/files/gtsam-tutorial.pdf