1. Project Overview

1.1 Purpose

This repository provides **open-source, simulation-based examples** of the algorithms in *Principles of Indoor Positioning and Indoor Navigation* (Chapters 2–8).

The repo is intended as a **companion to the book**: readers can read the equations and derivations in the text and then run concrete, minimal code examples that directly implement those equations on simulated datasets.

1.2 Target Users

- Master's students in navigation / robotics / geodesy / EE / CS.
- PhD students and early-career researchers entering indoor positioning.
- Engineers who want a **reference implementation** of classical algorithms before building production systems.

1.3 Goals

The repo should:

- Demonstrate key algorithms from Ch.2–8 on **small, reproducible simulations**:
    - **Ch.2:** coordinate systems & attitude.
    - **Ch.3:** LS/WLS, robust LS, KF/EKF/UKF/PF, FGO.
    - **Ch.4–7:** RF positioning, fingerprinting, PDR/sensors, SLAM.
    - **Ch.8:** practical sensor fusion.
- Provide **reproducible experiments** based on open simulation datasets.
- Offer **simple, inspectable reference implementations** rather than production systems.
- Support **per-chapter examples** that map back to the book.

New Goal: Equation-Level Traceability

For every important equation in the book that is implemented in code, there must be a **clear mapping**:

- **Book → Code**
    - For a given equation number (e.g. Eq. (3.12)), users can quickly locate:
        - The implementing function/class.
        - The tests and example notebooks that exercise it.

- **Code → Book**
    - From a function or class, users can see:
        - Which equation(s) it implements.
        - The chapter/section context.

Design constraints:

- Mapping must be:
    - **Searchable** via plain text (e.g. searching "Eq. (3.12)" in the repo).
    - **Maintainable** when new code is added.
    - Visible in **docstrings, comments, docs, and notebooks**.

1.4 Out of Scope

- Advanced techniques from Chapter 9 (crowdsourcing, collaborative, deep AI PDR, RIS) beyond small demos.
- Real-time deployment, mobile apps, or large-scale real-data pipelines.
- Full SLAM frameworks or high-performance mapping stacks.

---

2. Scope & Use Cases

2.1 Scope

The repo implements **simulation-first** reference versions of:

- Coordinate transforms (body/map/ENU/NED/LLH) and attitude representations.
- LS/WLS and robust LS; KF/EKF/UKF/PF; basic FGO wrappers.
- RF point positioning (TOA / two-way TOA / TDOA / AOA / RSS).
- Fingerprinting (Wi-Fi / magnetic / hybrid; deterministic & probabilistic).
- Proprioceptive sensor models (IMU, wheel odom, PDR, barometer, magnetometer).
- Minimal 2D SLAM & sensor fusion demos (Loosely vs Tightly coupled; observability).

2.2 Typical Use Cases

- “Run a notebook to reproduce a figure similar to chapter X.”
- “Swap KF ↔ FGO on the **same** simulated trajectory.”

- “Compare RF trilateration vs TDOA vs AOA in a toy floor.”
- “Compare k-NN fingerprinting vs a simple ML model on a synthetic RF map.”

This section sets **what problems the repo must solve**, not just what code exists.

---

3. High-Level Architecture

Repo layout:

```
ipin-examples/
  core/
    coords/
    estimators/
    rf/
    sensors/
    sim/
    eval/
  ch2_coords/
  ch3_estimators/
  ch4_rf_point_positioning/
  ch5_fingerprinting/
  ch6_dead_reckoning/
  ch7_slam/
  ch8_sensor_fusion/
  data/
    sim/
    real/    # optional, small demo logs only
  notebooks/
  docs/
  tools/
    check_equation_index.py
```

Directory roles:

- **core/**
  Reusable math & models used across chapters (never import from chX_.../):
  - coords/ – frames, LLH↔ECEF↔ENU, ENU↔NED, attitude conversions.
  - estimators/ – LS/WLS, robust LS, KF/EKF/UKF/PF, FGO wrappers.
  - rf/ – TOA/TDOA/AOA/RSS models, DOP utilities.
  - sensors/ – IMU, wheel odom, PDR, mag, barometer models.
  - sim/ – trajectory generators, scenario definitions, noise injection.
  - eval/ – error metrics, CDFs, NEES/NIS, DOP.
- **chX_.../**
  Thin chapter-specific examples:
  - Wiring code & scripts.
  - Plots and figure reproduction.
  - Minimal glue code between core/ and data.
- **data/sim/**
  Standardized **simulation datasets** (Section 5).
- **data/real/**
  Optional, very small demonstration logs (not the main focus).
- **notebooks/**
  One notebook (or small set) per chapter, referencing equations and pointing to core/ modules.
- **docs/**
  Markdown docs per chapter + equation_index.yml and usage docs.
- **tools/**
  CI/maintenance scripts, especially equation mapping checker.

---

4. Core Functional Requirements (Basic Functions)

For each core submodule, the design doc specifies:

- **Purpose**
- **Key data structures**

- **Required functions / classes** (with rough signatures)
- **Equation mapping** expectations (docstring conventions, index entries)
- **Owner**: navigation engineer vs software engineer.

4.1 core/coords

Purpose

- Implement coordinate and attitude foundations of Ch.2.

Basic functions

- Geodetic / ECEF / local frames:
    - llh_to_ecef(llh) -> np.ndarray
    - ecef_to_llh(ecef) -> np.ndarray
    - ecef_to_enu(ecef, ref_llh) -> np.ndarray
    - enu_to_ecef(enu, ref_llh) -> np.ndarray
    - enu_to_ned(enu) -> np.ndarray
    - ned_to_enu(ned) -> np.ndarray
- Attitude conversions:
    - rpy_to_rotmat(roll, pitch, yaw) -> np.ndarray
    - rotmat_to_rpy(R) -> Tuple[float, float, float]
    - quat_to_rotmat(q) -> np.ndarray
    - rotmat_to_quat(R) -> np.ndarray

Equation mapping

- All core transforms have docstrings like:
    - "Implements Eq. (2.x) in Chapter 2: ..."
- Each implemented equation appears in docs/equation_index.yml.

Unique tasks

- Support multiple map frames (multi-floor, building frames).
- Enforce consistent frame naming/metadata used by RF, PDR, SLAM.

4.2 core/estimators

Purpose

- Implement core estimation and filtering tools for Ch.3 and later.

Basic functions/classes

- LS & robust LS:
    - linear_least_squares(A, b) -> x # Eq. (3.12)
    - weighted_least_squares(A, b, W) -> x
    - Robust LS wrappers (Huber, Cauchy).
- Filters:
    - KalmanFilter
    - ExtendedKalmanFilter
    - UnscentedKalmanFilter
    - ParticleFilter
- FGO:
    - Factor interface:
        - class Factor: def evaluate(self, x) -> (residual, jacobian)
    - Solver interface for Gauss-Newton / LM.

Equation mapping

- Each key method (predict, update, LS solver, etc.) references the corresponding equations in Ch.3 (e.g. time update, covariance update, robust cost functions).

Unique tasks

- Provide **common APIs** so RF, PDR, SLAM examples can plug in the same estimators.

4.3 core/rf

Purpose

- RF signal measurement models from Ch.4–5.

Basic functions

- Ranging / angles:
    - toa_range(tx_pos, rx_pos, c, clock_bias)
    - two_way_toa_range(...)
    - tdoa_range_diff(anchor_i, anchor_j, state, c)

    - aoa_bearing(anchor_pos, state)
- RSS:
    - rss_pathloss(tx_power, distance, n, sigma_shadow)
- DOP:
    - compute_dop(geometry, weights=None) -> dict with HDOP/VDOP/PDOP.

Equation mapping

- Each measurement model documents which equations in Ch.4 it implements (e.g. path-loss, hyperbolic TDOA, AOA geometry).

Unique tasks

- Support multiple technologies (Wi-Fi, BLE, UWB, 5G) through parameterization rather than separate code.

4.4 core/sensors

Purpose

- Proprioceptive & environmental sensor models from Ch.6.

Basic functions

- IMU:
    - Continuous/discrete error models (bias, noise, RW).
    - Simple strapdown integration for 2D/3D.
- PDR:
    - Step detection, step length models, heading from IMU/mag.
- Environmental:
    - Barometer altitude, floor detection.
    - Magnetometer heading and magnetic fingerprint stubs.
- Wheel odometry:
    - Differential drive, steering model, noise models.

Equation mapping

- IMU propagation, error models, ZUPT updates etc. point to Ch.6 equations.

Unique tasks

- Define a **unified sensor packet** structure used across simulation and logs.

4.5 core/sim

Purpose

- Shared simulation tools for all examples.

Basic functions

- Trajectories:
    - generate_2d_trajectory(shape, params, dt, duration)
    - generate_3d_trajectory(...)
- Scenarios:
    - Rooms/floor plans, beacons/anchors, obstacles.
- Sensor noise injection:
    - add_imu_noise(traj, model_params)
    - simulate_rf_measurements(traj, anchors, rf_config)

Unique tasks

- Scenario configurations in JSON/YAML so all chapters share the same definitions.

4.6 core/eval

Purpose

- Evaluation utilities.

Basic functions

- Position errors:
    - RMSE, CDFs, histograms.
- Consistency metrics:
    - NEES, NIS calculators.
- DOP visualizers (reusing core/rf and core/coords).

Unique tasks

- Standardize evaluation APIs so results from different examples are comparable.

---

5. Simulation Datasets & Open Data

5.1 Principles

- All primary examples must run **only on data in data/sim/**.
- Datasets must be:
    - Small enough to clone and run quickly.
    - Reproducible (fixed seeds, recorded configs).
    - Clearly licensed (e.g. CC-BY-4.0) and documented in docs/data.md.

5.2 Dataset Families

Planned families under data/sim/:

- rf_2d_floor/
    - 2D floor map, anchor positions, true trajectories, TOA/TDOA/AOA/RSS measurements.
- wifi_fingerprint_grid/
    - Reference points on a grid, RSS vectors, test trajectories.
- pdr_corridor_walk/
    - IMU time series, reference path, step labels, floor labels.
- slam_lidar2d/
    - 2D occupancy grid, range-bearing measurements, ground truth poses.

For each dataset:

- File formats: CSV / NPZ / HDF5.
- Coordinate frames: clearly defined, link to core/coords.
- Ground truth semantics.

5.3 Optional Real Data

- data/real/ may contain **small** demo logs (not full research datasets) to show real-world quirks.
- Not a core dependency for running per-chapter examples.

---

6. Equation–to–Code Mapping Design

This section defines how equation mapping is implemented, enforced, and integrated with core/ and chapter modules.

6.1 Canonical Equation ID Format

- Equations are referenced using a **canonical string**:
    - Eq. (C.NN) for Chapter C, Equation NN.
    - If the book uses section-based numbering, adapt to Eq. (C.SS).

Examples

- Chapter 2, equation 5 → "Eq. (2.5)"
- Chapter 3, equation 12 → "Eq. (3.12)"

Rules

- This exact string must appear in:
    - Docstrings.
    - Comments (for line-level mapping).
    - Notebooks (markdown cells).
    - docs/equation_index.yml.

That way, a simple repo search for "Eq. (3.12)" finds all relevant code/tests.

6.2 Code Conventions (Docstrings & Comments)

Requirement

Every function/class that directly implements a book equation must mention it in its **docstring**.

Example: LS

```
def linear_least_squares(A: np.ndarray, b: np.ndarray) -> np.ndarray:
  """
  Solve the linear least squares problem x* = argmin ||Ax - b||_2.


  Implements Eq. (3.12) in Chapter 3 of the book:
    x* = (A^T A)^{-1} A^T b
  """
  ...
```

Example: EKF predict

```
def predict(self, u: np.ndarray) -> None:
  """
  Extended Kalman filter prediction step.

  Implements Eqs. (3.25)–(3.27) in Chapter 3:
    - State propagation: Eq. (3.25)
    - Jacobian: Eq. (3.26)
    - Covariance propagation: Eq. (3.27)
  """
  ...
```

Line-level comments

If a specific line encodes an equation:

```
# Eq. (3.27): P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k
P_pred = F @ P @ F.T + Q
```

Design rule

- Whole function = one equation (or block) → docstring reference.
- Single expression or key step = equation → inline comment with the exact Eq. (C.NN) reference.

6.3 Central Equation Index File

Create docs/equation_index.yml (or .json).

Purpose

From the book side, you can look up "Eq. (3.12)" and see:

- Which modules/objects implement it.
- Which tests/notebooks exercise it.

YAML structure (example)

```
- eq: "Eq. (2.3)"
  chapter: 2
  description: "ECEF to ENU coordinate transformation"
```

```
  files:
    - path: "core/coords/frames.py"
      object: "ecef_to_enu"
    - path: "ch2_coords/examples/coord_demo.py"
      object: "demo_ecef_to_enu"
  tests:
    - "tests/test_coords.py::test_ecef_enu_roundtrip"


- eq: "Eq. (3.12)"
  chapter: 3
  description: "Linear least squares closed form"
  files:
    - path: "core/estimators/least_squares.py"
      object: "linear_least_squares"
  notebooks:
    - "notebooks/ch3_estimators/ls_vs_ekf.ipynb"
```

Design choices

- One entry per implemented equation.
- Keys:
    - eq – canonical ID ("Eq. (3.12)").
    - chapter
    - description
    - files – list of {path, object}.
    - Optional: tests, notebooks, notes.

6.4 Notebooks & Docs Conventions

Notebooks

At the top of each chapter notebook, add an “Equations used” list, e.g.:

Equations in this notebook

- Eq. (3.12): Linear least squares solution.
- Eqs. (3.25)–(3.27): EKF prediction equations.

And reference equations in text:

We now implement the linear least squares estimator from Eq. (3.12).

Docs

In docs/ch3_estimators.md:

Equation map

- Eq. (3.12): `core/estimators/least_squares.py::linear_least_squares`
- Eqs. (3.25)–(3.27): `core/estimators/kalman.py::ExtendedKalmanFilter.predict`

This gives users three navigation routes:

1. Search "Eq. (3.12)" in the repo.
2. Check docs/equation_index.yml.
3. Read chapter docs listing mappings.

6.5 Tooling: Equation Index Checker Script

Add tools/check_equation_index.py:

Responsibilities:

- Parse docs/equation_index.yml.
- For every {path, object} entry:
    - Verify the module and object exist.
    - Verify the docstring or comments contain the mapped "Eq. (C.NN)".

Sketch (already provided):

- Use importlib to load modules.
- Inspect obj.__doc__.
- Fail CI if any mismatch.

Integrate in CI:

- Run pytest.
- Run python tools/check_equation_index.py.

6.6 Integration into Epics / Phases

For each epic, “definition of done” includes:

- Code & tests written.
- Example notebook added.
- **Equation mapping completed**:
    - Docstring references added.
    - equation_index.yml updated.
    - Notebook/docs list relevant equations.

Concretely:

- **Epic 1 – Coordinates (Ch.2)**
    - All transforms tagged with Eq. (2.x) where applicable.
- **Epic 2 – Estimators (Ch.3)**
    - LS, KF, EKF, UKF, PF, FGO all mapped to Ch.3 equations.
- **Epic 3 – RF positioning (Ch.4)**
    - TOA/TDOA/AOA/RSS and DOP models mapped.
- Subsequent epics (fingerprinting, PDR, SLAM, fusion) follow the same pattern.

---

7. Per-Chapter Example Modules (Unique Tasks)

For each chX_... folder, list:

- Example scripts / notebooks.
- Which core/ functions they exercise.
- Which equations they highlight.

7.1 ch2_coords/

Examples

- demo_frames.py
    - Build body/map/ENU/NED frames and print transforms.
- Simple plotting script to reproduce or echo Ch.2 figures.

Dependencies

- core.coords, core.sim, core.eval.

7.2 ch3_estimators/

Examples

- ls_vs_robust_ls.py
    - Outlier demo: LS vs Huber vs Cauchy.
- kf_vs_ekf_vs_ukf_vs_pf_vs_fgo.ipynb
    - 1D/2D toy system comparing estimator families.

Unique tasks

- Compare convergence, RMSE, computation time.
- Show qualitatively how different estimators behave under same conditions.

7.3 ch4_rf_point_positioning/

Examples

- TOA trilateration (LS/WLS).
- TDOA Chan & Fang algorithms.
- AOA OVE / PLE examples.

Dependencies

- core.coords, core.rf, core.estimators, core.eval.

7.4 ch5_fingerprinting/

Examples

- Synthetic grid map:
    - Generate RSS fingerprints from core.rf and core.sim.
- Deterministic methods:
    - k-NN, weighted k-NN.
- Probabilistic:
    - Naive Bayes / simple Gaussian likelihood.
- Optional:
    - Small MLP (e.g. 2–3 layers) as a “modern” baseline.

7.5 ch6_dead_reckoning/

Examples

- Foot-mounted PDR with ZUPT:
    - Walk along corridor, accumulate drift, correct with constraints.
- Vehicle odom + IMU:
    - Simple car-like motion on a map.

7.6 ch7_slam/

Examples

- Minimal 2D LiDAR / range-bearing SLAM:
    - Use FGO with pose + landmark factors.
- Loop closure toy example.

7.7 ch8_sensor_fusion/

Examples

- Loosely vs tightly coupled fusion examples:
    - GNSS+IMU style but adapted to indoor sensors.
- Observability demo:
    - Show drift/unobservability vs properly fused sensors.
- Calibration:
    - Simple intrinsic/extrinsic calibration toy.

For each chapter section, the design doc should separate:

- **Core functions reused** from core/.
- **Chapter-specific "unique tasks"** and plots.

---

8. Non-Functional Requirements

8.1 Language & Tooling

- Python 3.x.
- NumPy, SciPy, matplotlib, optional JAX/PyTorch for ML examples (kept light).
- Packaging via pyproject.toml / poetry or similar.

8.2 Quality

- Unit tests for all core functions (tests/).
- pytest-based CI.
- Equation index checker included in CI.

8.3 Performance

- All examples should run on a typical laptop in **minutes**, not hours.
- Datasets and notebooks: no GPU dependency required.

8.4 Documentation

- docs/:
    - Per-chapter docs (overview, how to run examples).
    - docs/equation_index.yml and usage instructions.
- README.md:
    - Explain structure, target audience, and how to map book ↔ repo.

---

9. Implementation Roadmap

Use epics as a roadmap for you + contributing engineers.

9.1 Epic 0 – Repo & Infrastructure

- Initialize repo structure.
- Set up:
    - core/, chX_.../, data/, notebooks/, docs/, tools/.
    - CI with pytest + equation checker.

9.2 Epic 1 – Coordinates (Ch.2)

- Implement core/coords.
- Add tests for:
    - LLH↔ECEF↔ENU round-trip.
    - ENU↔NED conversions.
- Implement ch2_coords examples.
- Add Ch.2 equations to equation_index.yml.

9.3 Epic 2 – Estimators (Ch.3)

- Implement LS/WLS + robust LS in core/estimators.
- Implement KF/EKF/UKF/PF skeletons.
- Add simple FGO wrapper.
- Unit tests (simple linear systems).
- ch3_estimators example notebook.
- Update equation_index.yml for Ch.3.

9.4 Epic 3 – RF Positioning (Ch.4)

- Implement measurement models and DOP in core/rf.
- Build RF simulation tools in core/sim.
- ch4_rf_point_positioning examples:
    - TOA, TDOA (Chan/Fang), AOA demos.
- Add Ch.4 equations to equation_index.yml.

9.5 Later Epics – Fingerprinting, Sensors, SLAM, Fusion

- Similar pattern:
    - Spec → core implementation → tests → examples → docs → equation mapping.

---

10. Working Effectively: SW vs Navigation Engineers

Throughout the design doc, separate:

- **Algorithm / modeling decisions** (navigation engineer):
    - Which equation(s) in the book to implement.
    - What approximations and parameters to use.
- **Implementation details** (software engineer):
    - File layout, function signatures, performance, tests, CI.

The equation mapping system bridges the two:

- Nav engineers can **check correctness** by tracing Eq. → code.
- SW engineers can **preserve traceability** by following docstring + index conventions and the equation checker.