COURSE CERTIFIED BY:
COURSE CERTIFIED THRU:

SY6301
Navigation and State Estimation

Juan “Silv” Jurado, Col, USAF, PhD
Permanent Professor and Head
Electrical and Computer Engineering

Terminal Learning Objectives

By the end of this course you should be able to:

- Apply foundational navigation concepts to air and space systems
- Construct and analyze navigation measurement and dynamic models (GPS, INS, AltNav)
- Implement and evaluate estimation methods (Least Squares, Kalman Filter, EKF)
- Assess navigation performance, integrity, and fault behavior using quantitative metrics
- Synthesize and communicate data-driven conclusions for navigation systems under test

Silv's ROEs

- These are my slides. Flag anything that could be sharper.
- We're going from zero to hero in 9 hours. Full understanding requires both attention and interaction. Stay engaged.
- Ask questions early and often. Slow me down anytime.
- I've been a TPS student myself, so I know you're constantly making priority calls. Side-task when you must, be considerate of others, and live with the consequences of what you choose to focus on.
- Slides are not meant to stand alone. Take notes actively.
- Use the course website to get deeper insights and reference materials.
- I'm on Slack 24/7 for questions or discussion.

Course Deliverables

Deliverable	Type	Weight	Due
Project PRR	Group	50%	Friday 12 June, 0800
Homework	Individual	50%	Sunday 14 June, 2359

SY6301 – Navigation and State Estimation

Block 1: Introduction, Frames, and Errors

Block Objectives

By the end of this block you should be able to:

- Explain what a navigation system estimates
- Distinguish between **truth**, **measurement**, and **estimate**
- Identify core frames used in navigation: **ECEF**, **NED**, **Body**
- Define position error and compute **horizontal** / **vertical** metrics
- Explain why frame consistency is required before statistics or integrity logic

What is Navigation?

- **Engineering definition:** Navigation estimates vehicle **state** relative to a chosen **reference frame**
- Typical state vector:

$$\hat{\mathbf{x}} = \begin{bmatrix} \text{position} \\ \text{velocity} \\ \text{attitude} \end{bmatrix}$$

- Inputs are imperfect: sensors, models, and assumptions contain noise and bias
- A navigation solution is therefore:

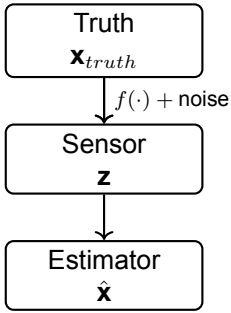
$$(\hat{\mathbf{x}},\ \mathbf{P})$$

  where **P** is the **state error covariance** (uncertainty)
- Together, they enable guidance, control, mission effectiveness, and safety

Truth vs Measurement vs Estimate

- **Truth**: $\mathbf{x}_{truth}$
- **Measurement**: $\mathbf{z} = f(\mathbf{x})$
- **Estimate**: $\hat{\mathbf{x}}$
- **Estimation error:**

$$\mathbf{e} = \hat{\mathbf{x}} - \mathbf{x}_{truth}$$



$$\mathbf{e} = \hat{\mathbf{x}} - \mathbf{x}_{truth}$$

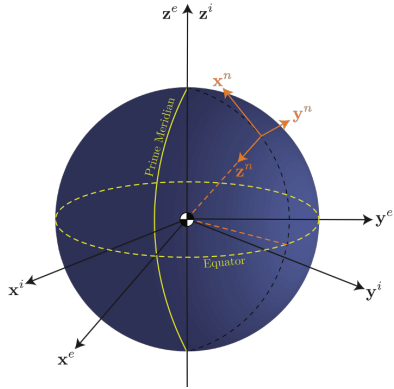
Why Frames Matter

- Suppose someone says: `error = [2, 1, -0.5]`
- You must ask:
  - In what frame? ECEF? NED? Body?
  - What are the units?
  - Where is the origin/reference defined?

Navigation computations only make sense with consistent frames.

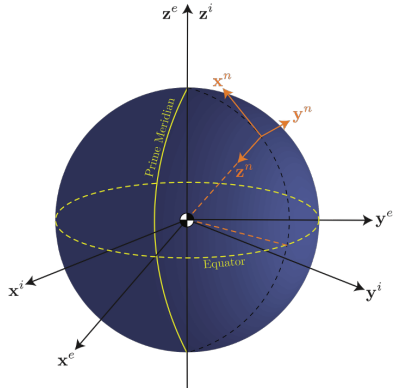
Frame 1: ECEF (Earth-Centered, Earth-Fixed)

- **Global reference:** Cartesian frame attached to the rotating Earth
- Axes:
  - X: toward lat $0°$, lon $0°$ (equator, Greenwich)
  - Y: toward lat $0°$, lon $90°$E (equator, east)
  - Z: toward North Pole (Earth spin axis)
- Origin at Earth's center of mass
- Natural for GNSS geometry and global trajectories
- Less intuitive for local error interpretation



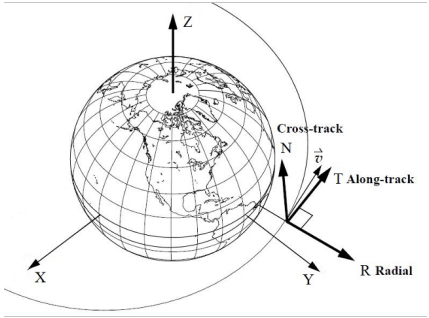
Frame 2: Local-Level NED (North-East-Down)

- **Local reference:** Tangent frame at a chosen reference LLH
- Axes:
  - N: north
  - E: east
  - D: down
- Assumes locally flat plane near the reference point
- Standard for aviation navigation and error metrics
- Great for plotting position error, velocity error, and covariances



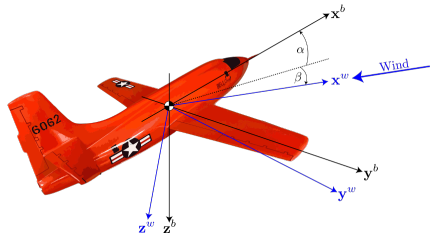
Frame 3: Radial-Transverse-Normal (RTN)

- **Local orbital reference:**
  Spacecraft-centered, oriented to position and velocity in inertial space
- Axes:
  - R: radial, outward from Earth's center
  - T: along-track, tangent to orbit in direction of motion
  - N: cross-track, normal to orbital plane (right-hand rule)
- Rotates with orbital motion; time-varying angular rate
- Standard for orbital relative motion (rendezvous, proximity ops)



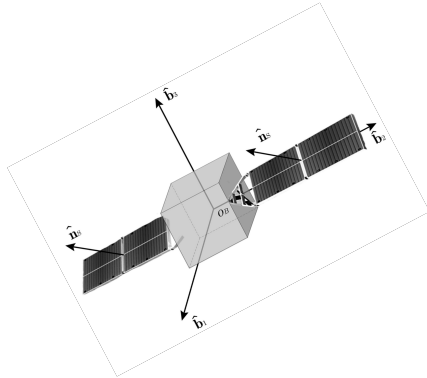
Frame 4: Aircraft Body Frame (Vehicle-Fixed, FRD)

- **Sensor reference:** Frame rigidly attached to the vehicle
- Axes:
  - X: forward
  - Y: right
  - Z: down
- IMU accelerometers measure specific force in body
- IMU gyroscopes measure body-to-NED attitude



Frame 5: Spacecraft Body Frame (sc-frame)

- **Vehicle-fixed:** Frame attached to the spacecraft structure; carries attitude, torque, and sensor pointing
- Axes (mission-dependent):
  - X: instrument boresight or nadir-pointing
  - Y: along solar array or side panel
  - Z: completes the right-hand rule
- Rotates and translates with the spacecraft
- Reference for star trackers, gyros, sun sensors, thrusters



Frame Transformations: Direction Cosine Matrices (DCMs)

- **Goal:** Express the *same physical vector* in a different frame
- **Convention:** DCM $\mathbf{C}_a^b$ transforms vectors from frame a to frame b

$$\mathbf{v}^b = \mathbf{C}_a^b \, \mathbf{v}^a$$

- **DCM properties:**
  - Orthonormal: $\mathbf{C}^\top \mathbf{C} = \mathbf{I}$
  - Inverse equals transpose:

$$\left(\mathbf{C}_a^b\right)^{-1} = \left(\mathbf{C}_a^b\right)^\top = \mathbf{C}_b^a$$

Rotation moves components, not the physical vector.

Frame Transformations: 2D DCM Example

- Consider two frames rotated by angle θ
- Components change, vector does not

$$\mathbf{v}^b = \mathbf{C}_a^b(\theta)\,\mathbf{v}^a$$

- 2D rotation matrix:

$$\mathbf{C}_a^b(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

- Key intuition:
  - Same arrow in space
  - Different numerical representation

Demo: 2D Direction Cosine Matrix

MATLAB version in `code/DcmExample.m`

Frame Transformations: 3D DCMs and Moving Points

- 3D generalization:

$$\mathbf{v}_b = \mathbf{C}_a^b \mathbf{v}_a, \qquad \mathbf{C}_a^b \in \mathbb{R}^{3\times 3}$$

- One parameterization: **Roll–Pitch–Yaw**

$$\mathbf{C}_a^b(\phi,\theta,\psi) = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi)$$

  - Order matters

- **Vectors vs Points:**
  - Vectors: rotation only
  - Points: rotation + translation

- Point transform:

$$\mathbf{p}_b = \mathbf{C}_a^b\,(\mathbf{p}_a - \mathbf{r}_{a\to b})$$

Error Definitions in NED

- Error vector can be defined in NED frame
- Intuitive for operational and test reporting
- Connects directly to targeting requirements

$$\mathbf{e}_{NED} = \begin{bmatrix} e_N \\ e_E \\ e_D \end{bmatrix}$$

$$e_H = \sqrt{e_N^2 + e_E^2}, \qquad e_V = |e_D|$$

Quick Exercise

$$\mathbf{p}_{truth} = \begin{bmatrix} 1000 \\ 500 \\ -50 \end{bmatrix}, \quad \hat{\mathbf{p}} = \begin{bmatrix} 1003 \\ 498 \\ -49 \end{bmatrix}$$

- Compute $\mathbf{e}_{NED}$
- Compute e_H and e_V
- If rotated into ECEF, what changes? What does not?

Demo: Target Location Error (TLE)

- **Mission goal:** Quantify targeting error from an airborne sensor
- Express both positions in the same frame:
  - Sensor-derived target LLH $\rightarrow$ ECEF $\rightarrow$ NED
  - True target LLH $\rightarrow$ ECEF $\rightarrow$ NED
- Compute the NED error vector: $\mathbf{e}_{NED} = \hat{\mathbf{p}}_{NED} - \mathbf{p}_{NED}$
- Reduce to operational metrics (e_H and e_V)
- **Interpretation:** Horizontal and vertical miss distance

Demo: Target Location Error

MATLAB version in `code/TargetLocationError.m`

Wrap-Up

- Navigation estimates vehicle **state** relative to a chosen **reference frame**, paired with a **covariance P** that quantifies uncertainty
- **Truth**, **measurement**, and **estimate** are distinct quantities; the estimation error is $\mathbf{e} = \hat{\mathbf{x}} - \mathbf{x}_{truth}$
- Three core frames: **ECEF** (global Cartesian), **NED** (local-level tangent), **Body** (vehicle-fixed)
- **DCMs** transform vectors between frames; points additionally require a translation
- Position error reduces to operational metrics: $e_H = \sqrt{e_N^2 + e_E^2}$ and $e_V = |e_D|$
- Frame consistency is a **prerequisite** for any statistics or integrity logic that follows

Next block: where do those position errors come from? Inertial sensors integrate small errors into growing drift.

SY6301 – Navigation and State Estimation

Block 2: Inertial Error Growth and State Propagation

Block Objectives

By the end of this block you should be able to:

- Explain why inertial navigation errors **grow over time**
- Understand how small sensor errors propagate into **velocity and position**
- Identify common error models used in navigation systems
- Describe the concept of **state propagation**
- Interpret inertial drift using simple error metrics

What an IMU Measures

- Inertial Measurement Unit (IMU) provides:
  - **Specific force** from accelerometers
  - **Angular rate** from gyroscopes
- Measurements are expressed in the **body frame**
- Inertial navigation algorithms integrate these measurements to estimate:
  - Velocity
  - Position
  - Attitude

The Inertial Error Integration Chain

- Small sensor errors propagate through the “mechanization” equations
- Accelerometer error produces:
  - Velocity error (single integration)
  - Position error (double integration)
- Gyroscope error produces:
  - Attitude error
  - Incorrect projection of acceleration into NED frame
  - Further velocity and position error

Inertial drift is the result of integrating imperfect measurements.

Common Inertial Error Types

- **Bias**
  - Constant offset in a measurement
  - Produces linearly growing velocity error
- **Random noise**
  - Sample-to-sample variation
  - Produces random walk behavior
- **Drift**
  - Accumulated navigation error over time

Simple Error Models

- To estimate navigation performance, we use simplified models
- Example: Velocity bias random walk

$$b_{v,k+1} = b_{v,k} + w_k$$

- Example: Correlated noise (Gauss-Markov)

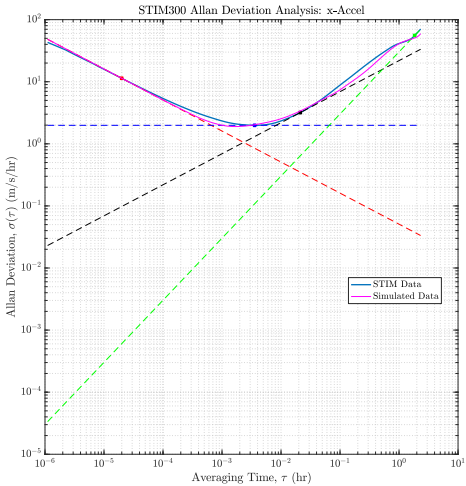
$$x_{k+1} = e^{-dt/T} x_k + w_k$$

- These models capture the **statistics** of sensor behavior

Error models capture statistics without modeling every physical detail.

Allan Variance: Characterizing IMU Errors on the Ground

- **Ground test, not flight test:** long static IMU run, typically ~ 6 hours
- Separates the noise processes in the IMU sensors
- **Procedure:** compute Allan deviation $\sigma(\tau)$ over cluster times τ and read slopes off the log-log plot
  - $-1/2$: white noise (angle/vel RW)
  - 0: bias instability
  - $+1/2$: rate random walk
- **Why we care:** numerical values feed straight into the Kalman filter as **Q** and as bias dynamics (random walk or Gauss-Markov)



Navigation State

- Navigation algorithms estimate a **state vector**

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{a} \end{bmatrix}$$

- Where
  - **p** = position (3×1)
  - **v** = velocity (3×1)
  - **a** = attitude between body and NED frames (3×1)
- The estimator also maintains an **uncertainty estimate**:

$$\mathbf{P} = \text{covariance of the state estimate } (9 \times 9)$$

State Propagation

- Navigation algorithms repeatedly **propagate** the state forward in time

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k$$

- Where
  - $\mathbf{x}_k$ = current state
  - $\mathbf{u}_k$ = IMU sensor measurements
  - $\mathbf{w}_k$ = process noise
- The estimator must also propagate the **covariance**

$$\mathbf{P}_{k+1} = \mathbf{F}_k \mathbf{P}_k \mathbf{F}_k^\top + \mathbf{Q}_k$$

- This describes how **uncertainty grows over time** as we propagate

Quick Exercise

Assume a constant accelerometer bias produces a constant velocity error.

- What happens to position error over time?
- Does the error grow linearly or quadratically?
- What happens if the system receives periodic corrections?

Why Inertial Drift Tests Are Long

- Inertial navigation errors evolve over time
- Horizontal inertial errors often exhibit a natural oscillation called the **Schuler cycle**

$$T_{Schuler} \approx 84.4 \text{ minutes}$$

- To properly characterize inertial drift, test runs should cover at least one full cycle
- Practical rule of thumb in flight test:

Inertial drift runs should be at least 85 minutes

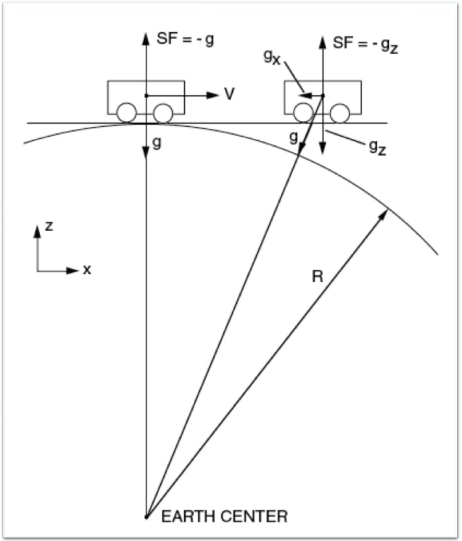
Good inertial testing requires observing a full Schuler cycle.

Why the Schuler Cycle Happens

- Consider a small **attitude error** in the navigation solution
- Gravity is now projected slightly into the horizontal plane

$$a_H \approx g\,\theta$$

- This produces a horizontal acceleration error
- The navigation solution begins to drift horizontally
- The system eventually **overshoots and corrects**, producing an oscillation



Demo: Inertial Drift

- Simulated inertial navigation with no external aiding
- Observe horizontal error growth
- Compute drift slope over time
- Interpret drift in operational units (NM/hr)

Demo: Inertial Drift

MATLAB version in `code/InertialDriftDemo.m`

Wrap-Up

- Inertial navigation integrates sensor measurements to estimate motion
- Small sensor errors accumulate through integration
- Drift is an unavoidable property of inertial systems without external updates
- Navigation algorithms propagate a state forward in time using motion models
- Schuler cycle drives required length of flight tests for inertial-only requirements

Next block: if inertial drift is unavoidable, can a second sensor bound it? $\rightarrow$ **optimal fusion**.

SY6301 – Navigation and State Estimation

Block 3: Optimal Fusion

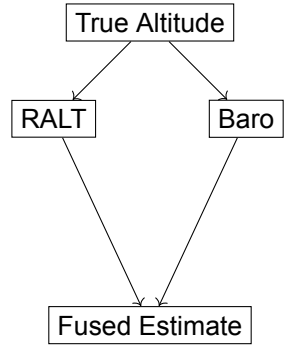
Block Objectives

By the end of this block you should be able to:

- Explain why combining sensors improves navigation estimates
- Compute a fused estimate using **weighted averaging**
- Understand why optimal weights depend on **measurement uncertainty**
- Compute the **variance of a fused estimate**
- Recognize that the **Kalman filter is recursive optimal fusion**

Why Fuse Sensors?

- Navigation systems often measure the same quantity using **multiple sensors**
- Example: altitude measurement
  - Radar altimeter: 100 ± 2 m
  - Barometric altitude: 100 ± 10 m
- Question: what altitude estimate should we use?
  - Choose one sensor?
  - Average them?
  - Something smarter?



Simple Averaging

Suppose two sensors measure the same quantity

$$z_1, \qquad z_2$$

A simple approach is the average

$$\hat{x} = \frac{z_1 + z_2}{2}$$

- Works if both sensors have **similar uncertainty**
- But what if

$$\sigma_1 = 1, \qquad \sigma_2 = 10$$

- Should both sensors influence the estimate equally?

Weighted Averaging

Instead of a simple average we can assign **weights**

$$\hat{x} = w_1 z_1 + w_2 z_2$$

subject to

$$w_1 + w_2 = 1$$

- Weights determine how much each sensor influences the estimate
- Sensors with lower uncertainty **should** receive a larger weight

Optimal Fusion: Problem Setup

Start with a weighted estimate

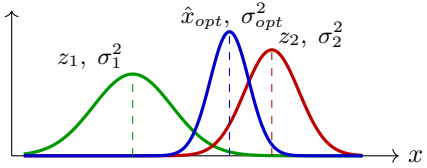
$$\hat{x} = w_1 z_1 + w_2 z_2$$

Assume measurement variances

$$Var(z_1) = \sigma_1^2, \qquad Var(z_2) = \sigma_2^2$$

The variance of the fused estimate is

$$Var(\hat{x}) = w_1^2 \sigma_1^2 + w_2^2 \sigma_2^2$$



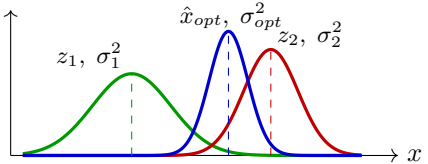
Optimal Fusion: Problem Setup

Goal: choose w_1, w_2 to minimize

$$Var(\hat{x}) = w_1^2\sigma_1^2 + w_2^2\sigma_2^2$$

subject to

$$w_1 + w_2 = 1$$



Solving the minimum-variance problem yields

$$w_1 = \frac{1/\sigma_1^2}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}}, \qquad w_2 = \frac{1/\sigma_2^2}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}}$$

Optimal Fusion Result

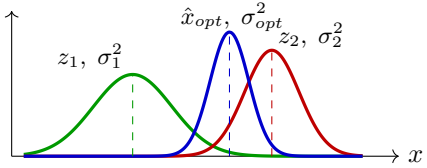
The optimal fused estimate becomes

$$\hat{x}_{opt} = \frac{\frac{z_1}{\sigma_1^2} + \frac{z_2}{\sigma_2^2}}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}}$$

The variance of the fused estimate is

$$\sigma_{opt}^2 = \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} \right)^{-1}$$

- Sensors with lower variance receive higher weight
- The fused estimate has lower uncertainty than either sensor alone



Optimal fusion reduces estimation uncertainty.

Information Interpretation

Define **information**

$$I = \frac{1}{\sigma^2}$$

Then the fused estimate becomes

$$I_{total} = I_1 + I_2$$

Interpretation:

- Each sensor contributes **information**
- Total information is the **sum of individual information**

Quick Exercise (2 minutes)

Two sensors measure the same quantity.

$$z_1 = 100, \qquad \sigma_1 = 2$$

$$z_2 = 110, \qquad \sigma_2 = 6$$

Use optimal fusion

$$\hat{x}_{opt} = \frac{\frac{z_1}{\sigma_1^2} + \frac{z_2}{\sigma_2^2}}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}}$$

- Compute the fused estimate $\hat{x}$
- Which sensor influences the estimate more?

Demo: Sensor Fusion

Simulate two noisy sensors

- Sensor 1: small noise
- Sensor 2: large noise

Fuse them using inverse-variance weighting

$$\hat{x} = \frac{\frac{z_1}{\sigma_1^2} + \frac{z_2}{\sigma_2^2}}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}}$$

Observe how the fused estimate reduces uncertainty.

Demo: Optimal Fusion

MATLAB version in `code/OptimalFusionDemo.m`

From Fusion to Kalman Filtering

Sensor fusion above assumes all measurements arrive **at once.**

Navigation systems instead process measurements **sequentially:**

$$\text{predict} \rightarrow \text{measure} \rightarrow \text{update}$$

This leads to the recursive update

$$\hat{x}_{new} = \hat{x}_{prior} + K(z - \hat{x}_{prior})$$

where

$$K = \frac{P}{P + R} \qquad \text{“Optimal Weight / Kalman Gain”}$$

- P = prior ($\hat{x}_{prior}$) uncertainty
- R = measurement (z) uncertainty

Wrap-Up

- Multiple sensors can be combined to improve navigation estimates
- Optimal fusion weights measurements using **inverse variance**
- Fusion reduces estimate uncertainty
- Recursive optimal fusion leads to the **Kalman filter**

Next block: fuse repeatedly as new measurements arrive → **recursive (Kalman) fusion**.

SY6301 – Navigation and State Estimation

Block 4: Scalar Kalman Filter

Block Objectives

By the end of this block you should be able to:

- Explain how the Kalman filter performs **recursive optimal fusion**
- Understand the roles of **prediction**, **measurement**, and **innovation**
- Interpret the **Kalman gain** as an uncertainty-weighted trust factor
- Understand how **state and covariance** are updated each step
- Prepare for multi-state navigation filters in the next lesson

Recursive Estimation Problem

Navigation systems operate **continuously over time**.

At each time step we have:

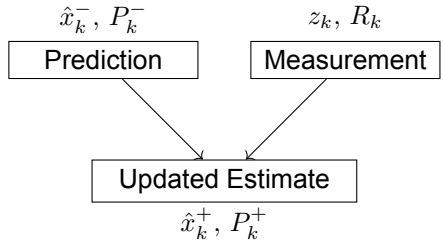
- A **prediction** and it's **uncertainty**:

$$\hat{x}_k^-, \quad P_k^-$$

- A **measurement** and it's **uncertainty**:

$$z_k, \quad R_k$$

Goal: Fuse prediction and measurement **optimally** to produce: $\hat{x}_k^+, \quad P_k^+$



Kalman Filter Update

The Kalman filter updates the estimate using

$$\hat{x}_k^+ = \hat{x}_k^- + K_k \underbrace{\left(z_k - \hat{x}_k^-\right)}_{\text{innovation}}$$

Where

- $\hat{x}_k^-$ = predicted state
- z_k = measurement
- $z_k - \hat{x}_k^-$ = innovation
- K_k = Kalman gain

Innovation

The **innovation** measures the difference between prediction and measurement.

$$\nu_k = z_k - \hat{x}_k^-$$

Interpretation:

- Small innovation $\rightarrow$ prediction was good
- Large innovation $\rightarrow$ measurement strongly corrects the estimate

Innovation measures the surprise in the measurement.

Kalman Gain

The scalar Kalman gain is

$$K_k = \frac{P_k^-}{P_k^- + R_k}$$

Where

- P_k^- = prediction uncertainty
- R_k = measurement variance

Interpretation:

- Large $R \rightarrow$ measurement noisy $\rightarrow$ small gain
- Large $P^- \rightarrow$ prediction uncertain $\rightarrow$ large gain

Kalman gain determines how much we trust the measurement.

Covariance Update

After incorporating the measurement

$$P_k^+ = (1 - K_k)P_k^-$$

Meaning:

- Uncertainty **decreases** after measurement update
- The filter estimates both the **state** and its **uncertainty**

Prediction Step

Before receiving the next measurement, we propagate the estimate forward.
State prediction:

$$\hat{x}_{k+1}^{-} = F\hat{x}_{k}^{+}$$

Covariance prediction:

$$P_{k+1}^{-} = FP_{k}^{+}F^{T} + Q_{k}$$

Where

- F = state transition model
- Q = process noise covariance

Update from previous time (k) becomes the prior for next time $(k+1)$

Scalar Kalman Filter Loop

Prediction: $k \rightarrow k+1$

$$\hat{x}_{k+1}^{-} = F\hat{x}_{k}^{+}$$

$$P_{k+1}^{-} = FP_{k}^{+}F^{T} + Q$$

Update: $- \rightarrow +$

$$K_{k+1} = \frac{P_{k+1}^{-}}{P_{k+1}^{-} + R}$$

$$\hat{x}_{k+1}^{+} = \hat{x}_{k+1}^{-} + K_{k+1}\left(z_{k+1} - \hat{x}_{k+1}^{-}\right)$$

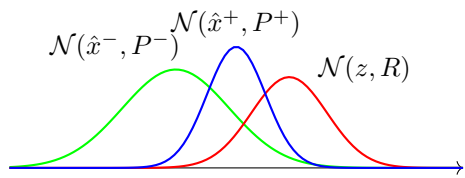
$$P_{k+1}^{+} = (1 - K_{k+1})P_{k+1}^{-}$$

Gaussian Interpretation

Prediction and measurement each provide information.

Kalman filtering fuses them to produce a posterior estimate.

- $\mathcal{N}(\hat{x}^-, P^-) \rightarrow$ Prior distribution
- $\mathcal{N}(z, R) \rightarrow$ Measurement distribution
- $\mathcal{N}(\hat{x}^+, P^+) \rightarrow$ Posterior distribution



Quick Exercise

Prediction

$$\hat{x}^- = 1000 \text{ m}$$

$$P^- = 25 \text{ m}^2$$

Measurement

$$z = 1010 \text{ m}$$

$$R = 9 \text{ m}^2$$

Compute

- Kalman gain
- Updated estimate (x^+) and variance (P^+)

Demo: Scalar Kalman Filter

Scenario: A train moves along a track at approximately constant velocity.

Simulation

- True train position evolves over time
- A sensor provides **noisy position measurements**
- A scalar Kalman filter estimates the train position

Key observations

- The filter smooths noisy measurements
- The estimate converges toward the true trajectory
- The covariance decreases as confidence grows

Demo: Scalar Kalman Filter

MATLAB version in `code/KalmanFilterScalar.m`

Wrap-Up

Key ideas

- Kalman filtering performs **recursive optimal fusion**
- The **innovation** measures the difference between prediction and measurement
- The **Kalman gain** determines how much the measurement influences the estimate
- The filter updates both the **state** and its **uncertainty**

Next block

- Multi-state Kalman filters for navigation systems
- Position, velocity, and attitude estimation (9-dimensional)

SY6301 – Navigation and State Estimation

Block 5: Multi-State Kalman Filter for Navigation

Block Objectives

By the end of this block you should be able to:

- Why navigation problems require **multi-state estimation**
- How the scalar Kalman filter generalizes to **vector states**
- The role of the **state transition matrix F**
- The role of the **measurement matrix H**
- How navigation filters simultaneously estimate **2D position and velocity**

Navigation filters estimate multiple states simultaneously

Why Multiple States?

The scalar Kalman filter estimated a **single variable**.

Navigation systems must estimate many variables simultaneously:

- Position
- Velocity
- Attitude
- Sensor biases
- Clock errors

The Kalman filter propagates and updates a state vector

State Vector Representation

Instead of estimating a single scalar, we estimate a **vector of states**. For a 2D tracking problem:

$$\mathbf{x}_k = \begin{bmatrix} p_{x,k} \\ p_{y,k} \\ v_{x,k} \\ v_{y,k} \end{bmatrix}$$

Where

- $p_{x,k},\ p_{y,k}$ = position in x and y at time step k
- $v_{x,k},\ v_{y,k}$ = velocity in x and y at time step k

State Dynamics Model

Assume constant-velocity motion in both axes.

$$p_{x,k+1} = p_{x,k} + v_{x,k}\Delta t$$

$$p_{y,k+1} = p_{y,k} + v_{y,k}\Delta t$$

$$v_{x,k+1} = v_{x,k}, \quad v_{y,k+1} = v_{y,k}$$

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In matrix form:

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k$$

Process Noise

- Real systems deviate from constant-velocity motion
- Unknown accelerations inject uncertainty into p_x, p_y, v_x, v_y
- Modeled as independent noise σ_a^2 on each axis

$$\mathbf{Q} = \sigma_a^2 \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & \Delta t^2 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & \Delta t^2 \end{bmatrix}$$

Process noise models uncertainty in the motion model

Measurement Model

Suppose the sensor (e.g., GPS) measures **2D position only**.

The measurement equation is:

$$\mathbf{z}_k = \begin{bmatrix} p_{x,k} \\ p_{y,k} \end{bmatrix} + \boldsymbol{\nu}_k$$

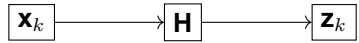
In matrix form:

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \boldsymbol{\nu}_k$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

R is now a 2×2 matrix:

$$\mathbf{R} = \begin{bmatrix} \sigma^2_{p_x} & 0 \\ 0 & \sigma^2_{p_y} \end{bmatrix}$$



Multi-State Kalman Filter Equations

Propagate:

$$\hat{\mathbf{x}}_{k+1}^{-} = \mathbf{F}\hat{\mathbf{x}}_{k}^{+}$$

$$\mathbf{P}_{k+1}^{-} = \mathbf{F}\mathbf{P}_{k}^{+}\mathbf{F}^{T} + \mathbf{Q}$$

Update:

$$\mathbf{K}_{k} = \mathbf{P}_{k}^{-}\mathbf{H}^{T}\left(\mathbf{H}\mathbf{P}_{k}^{-}\mathbf{H}^{T} + \mathbf{R}\right)^{-1}$$

$$\hat{\mathbf{x}}_{k}^{+} = \hat{\mathbf{x}}_{k}^{-} + \mathbf{K}_{k}\left(\mathbf{z}_{k} - \mathbf{H}\hat{\mathbf{x}}_{k}^{-}\right)$$

$$\mathbf{P}_{k}^{+} = \left(\mathbf{I} - \mathbf{K}_{k}\mathbf{H}\right)\mathbf{P}_{k}^{-}$$

Note: $\mathbf{R}$ is now a 2×2 matrix and $\mathbf{K}_{k}$ is 4×2.

Covariance Matrix Interpretation

The covariance matrix **P** describes **uncertainty in each state** and correlations between states.

$$\mathbf{P} = \begin{bmatrix} \sigma_{p_x}^2 & \sigma_{p_x p_y} & \sigma_{p_x v_x} & \sigma_{p_x v_y} \\ \sigma_{p_y p_x} & \sigma_{p_y}^2 & \sigma_{p_y v_x} & \sigma_{p_y v_y} \\ \sigma_{v_x p_x} & \sigma_{v_x p_y} & \sigma_{v_x}^2 & \sigma_{v_x v_y} \\ \sigma_{v_y p_x} & \sigma_{v_y p_y} & \sigma_{v_y v_x} & \sigma_{v_y}^2 \end{bmatrix}$$

Interpretation:

- Diagonal terms = uncertainty in each state (p_x, p_y, v_x, v_y)
- Off-diagonal terms = correlation between state errors (e.g., x-position and x-velocity)
- x and y cross-terms are zero for independent axes

Navigation Example

Operational navigation filters estimate many states simultaneously.

Example: **INS/GPS 15-state navigation filter**

$$\mathbf{x}_k = \left[\begin{array}{ccc|ccc|ccc|ccc|ccc} x & y & z & v_x & v_y & v_z & \phi & \theta & \psi & b_{ax} & b_{ay} & b_{az} & b_{gx} & b_{gy} & b_{gz} \end{array}\right]^T$$

Where:

- $(x,\ y,\ z)$ = position
- $(v_x,\ v_y,\ v_z)$ = velocity
- $(\phi,\ \theta,\ \psi)$ = attitude
- $\mathbf{b}_a$ = accelerometer biases
- $\mathbf{b}_g$ = gyroscope biases

Demo: 4-State Navigation KF

We will simulate:

- A vehicle moving in 2D with approximately constant velocity
- Noisy GPS position measurements in x and y
- A 4-state Kalman filter

The filter simultaneously estimates:

- 2D position: p_x, p_y
- 2D velocity: v_x, v_y

Demo: 4-State Kalman Filter

MATLAB version in `code/KalmanFilter4D.m`

Wrap-Up

- States form the vector $\hat{\mathbf{x}} = [p_x,\ p_y,\ v_x,\ v_y]^T$
- The 4×4 motion model is defined by **F**
- Measurements are mapped by the 2×4 matrix **H**
- Uncertainty is captured by the 4×4 covariance matrix **P**
- Measurement noise is now the 2×2 matrix **R**

Next block: measurements have been linear so far. GPS pseudoranges are not. $\rightarrow$ **GPS fundamentals**.

Navigation filters estimate PVA, sensor errors, and more!

SY6301 – Navigation and State Estimation

Block 6: GPS Fundamentals for Navigation

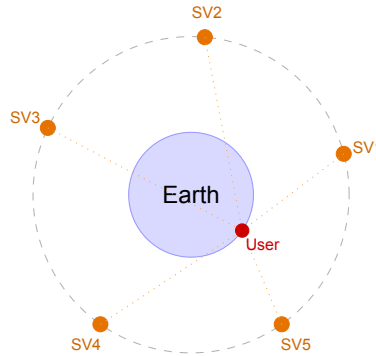
Block Objectives

By the end of this block you should be able to:

- Describe the GPS constellation and signal architecture
- Explain how a **pseudorange** is formed and why it is *pseudo*
- Formulate GPS positioning as a **nonlinear least-squares problem**
- Explain how satellite geometry amplifies pseudorange noise into position error
- Identify the four dominant GPS error sources and their typical magnitudes
- Interpret GPS accuracy specifications (CEP, 1σ, 2DRMS)

The GPS Constellation

- 24+ satellites in medium Earth orbit
  - Altitude ≈20,200 km Period ≈12 hours
- ≥ 4 satellites in view from anywhere on Earth at any time
- GPS is one of several constellations; collectively called **GNSS** (also GLONASS, Galileo, BeiDou)
- Each satellite broadcasts its precise position and an atomic timestamp



How GPS Measures Distance: The Pseudorange

- Satellite transmits a PRN code at known time t_{tx}; receiver records arrival at t_{rx}
- Apparent range: $c \cdot (t_{\text{rx}} - t_{\text{tx}})$
- **Problem:** receiver clock not synchronized with satellite $\rightarrow$ unknown bias δt_u corrupts every measurement

The result is a **pseudorange**:

$$\tilde{\rho}_i = \underbrace{\|\mathbf{r}_u - \mathbf{r}^{(i)}\|}_{\text{true range}} + c\,\delta t_u + \epsilon_i$$

This is a **nonlinear function** of user position $\mathbf{r}_u$

Pseudorange = true range + clock bias + noise

GPS as a Nonlinear Estimation Problem

- Four unknowns: $[x,\ y,\ z,\ \delta t_u]$
- Each satellite contributes one equation:

$$\tilde{\rho}_i = h_i(\mathbf{x}) + \epsilon_i, \qquad h_i(\mathbf{x}) = \|\mathbf{r}_u - \mathbf{r}^{(i)}\| + c\,\delta t_u$$

- Need ≥ 4 satellites for a unique solution
- Cannot be solved in closed form — solved iteratively (Newton-Raphson, Levenberg-Marquardt, MATLAB `fitnlm`)

GPS positioning is a nonlinear least-squares problem solved iteratively

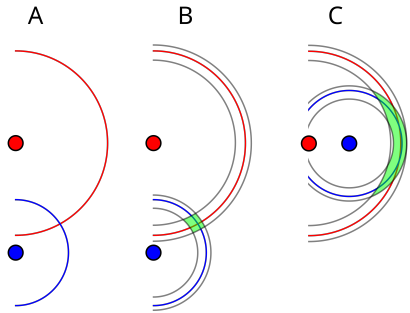
How Satellite Geometry Affects Accuracy

Each pseudorange defines a **sphere** centered on a satellite.

Position = intersection of spheres.

- **Well-spread:** spheres cross at a sharp angle → tight intersection → small uncertainty
- **Clustered:** spheres nearly parallel → poorly constrained → large uncertainty

Same σ_ρ — very different position accuracy



Dilution of Precision (DOP)

DOP quantifies the geometry effect:

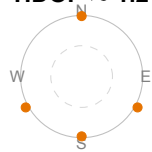
$$\sigma_{\text{position}} = \text{DOP} \times \sigma_\rho$$

Variants: **HDOP**, **VDOP**, **PDOP**, **TDOP**

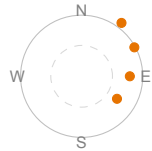
- HDOP ≈ 1–2: good geometry
- HDOP > 4: degraded accuracy

DOP is a **property of geometry only**

HDOP ≈ 1.2



HDOP ≈ 4.5



Geometry matters as much as pseudorange noise

GPS Error Sources

Four dominant contributors to pseudorange noise σ_ρ:

Source	Typical Magnitude	Mitigation
Ionospheric delay	1–5 m	Dual-frequency; model correction
Tropospheric delay	0.5–2 m	Saastamoinen model
Multipath	0.5–3 m	Antenna placement; signal processing
Receiver noise	0.1–0.3 m	Hardware quality

Lumped result: $\sigma_\rho \approx 3$–5 m (L1 SPS) $\longrightarrow$ sets diagonal of **R** in the Kalman filter

GPS accuracy is bounded by physics, not just hardware

GPS Limitations

- **Jamming:** signal at Earth's surface is very weak (≈ -130 dBm) — easily overpowered by a small transmitter
- **Spoofing:** counterfeit signals mislead the receiver; a growing and accessible threat
- **Denied environments:** indoors, underground, urban canyons, heavy foliage
- **Vertical accuracy:** always worse than horizontal (VDOP $>$ HDOP due to geometry)

GPS works until it doesn't — integration with INS fills the gaps

Quick Exercise

A GPS receiver tracks satellites with $\sigma_\rho = 3$ m.

Scenario A: Satellites well spread $\longrightarrow$ HDOP = 1.5

Scenario B: Satellites clustered $\longrightarrow$ HDOP = 4.0

- Compute horizontal 1σ position accuracy for each scenario
- A solar storm doubles σ_ρ to 6 m. What happens in each scenario?

$$\sigma_{\text{pos}} = \text{HDOP} \times \sigma_\rho$$

Demo: GPS Positioning

Scenario: 2D GPS positioning from pseudorange measurements

Simulation

- Simulate pseudoranges with realistic noise ($\sigma_\rho = 3$ m)
- Solve for position using nonlinear least-squares (Gauss-Newton)

Key observations

- Run with good geometry vs. poor geometry
- Visualize position uncertainty ellipses from `fitnlm` covariance
- Compare to DOP $\times\, \sigma_\rho$ prediction

Demo: GPS Positioning

MATLAB version in `code/GPSPositioningDemo.m`

Wrap-Up

- GPS measures **pseudoranges** corrupted by clock bias — not direct position
- Positioning requires ≥ 4 satellites to solve for $[x,\ y,\ z,\ \delta t_u]$ — a **nonlinear** problem
- Satellite geometry (**DOP**) amplifies pseudorange error: $\sigma_{\text{pos}} = \text{DOP} \times \sigma_\rho$
- Error sources (ionosphere, troposphere, multipath) set σ_ρ and the **R** matrix
- GPS has operational limits — inertial integration fills the gaps

Next block: $h_i(\mathbf{x})$ is nonlinear — how do we feed GPS into a KF built for linear measurements? $\rightarrow$ **Extended Kalman Filter**

Next block: extending the KF to handle nonlinear sensors

SY6301 – Navigation and State Estimation

Block 7: Extended Kalman Filter for GPS

Block Objectives

By the end of this block you should be able to:

- Distinguish between **loose coupling** and **tight coupling** GPS/INS integration
- Apply a standard Kalman filter update when GPS provides a **linear position measurement**
- Explain why pseudorange measurements require an **Extended Kalman Filter**
- Compute the **pseudorange Jacobian H** from geometry
- Trace through one complete **EKF predict-and-update cycle** with GPS pseudoranges

Two Ways to Fuse GPS into a Filter

Loose Coupling

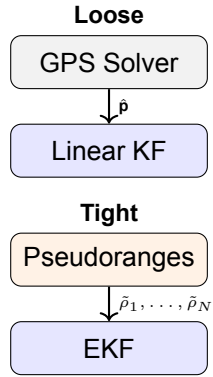
GPS solver runs first; filter receives solved position $\hat{\mathbf{p}}$

- Measurement: $\mathbf{z} = \hat{\mathbf{p}} + \boldsymbol{\nu}$
- Observation matrix: $\mathbf{H} = [\mathbf{I} \mid \mathbf{0}]$ picks off positions
- Standard linear KF update — no extra math

Tight Coupling

Filter ingests raw pseudoranges directly

- Measurement per satellite: $\tilde{\rho}_i = \|\mathbf{p} - \mathbf{s}^{(i)}\| + b$
- Nonlinear in $\mathbf{x}$ — requires linearization
- Extended Kalman Filter (EKF)



Loose Coupling: GPS as a Position Sensor

GPS solver outputs solved position $\hat{\mathbf{p}}$ — the filter treats it as a noisy measurement

$$\mathbf{z}_k = \hat{\mathbf{p}}_k + \boldsymbol{\nu}_k, \qquad \boldsymbol{\nu}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$$

For our **2D example** (state $= [p_x,\ p_y,\ v_x,\ v_y]^T$ from Block 5):

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \qquad \text{(same } \mathbf{H} \text{ as Block 5)}$$

In **3D** the state grows to $[p_x, p_y, p_z, v_x, v_y, v_z]^T$ and $\mathbf{H} = [\mathbf{I}_3 \mid \mathbf{0}_{3\times3}]$ — same idea, bigger matrix

R is set from GPS accuracy spec — $\sigma^2_{GPS} \approx (\text{HDOP} \times \sigma_\rho)^2$

Tight Coupling: The Challenge

Each satellite contributes one pseudorange measurement

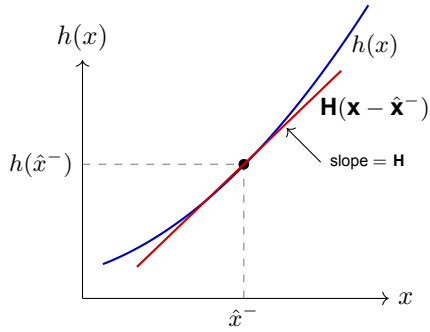
$$\tilde{\rho}_i = \underbrace{\|\mathbf{p} - \mathbf{s}^{(i)}\|}_{\text{nonlinear in } \mathbf{p}} + b + \epsilon_i$$

- The standard KF update **requires** a linear model $\mathbf{z} = \mathbf{H}\mathbf{x} + \boldsymbol{\nu}$
- There is no matrix **H** that exactly represents a range measurement
- **Solution:** linearize $h_i(\mathbf{x})$ around the current predicted state $\hat{\mathbf{x}}^-$

EKF Key Idea: Linearization

Replace the nonlinear function with its **first-order Taylor expansion**:

$$h(\mathbf{x}) \approx h(\hat{\mathbf{x}}^-) + \underbrace{\left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}^-}}_{\mathbf{H}} (\mathbf{x} - \hat{\mathbf{x}}^-)$$



The **innovation** uses the nonlinear function:

$$r = \tilde{\rho} - h(\hat{\mathbf{x}}^-)$$

Then the standard KF equations apply with $\mathbf{H}$ evaluated at $\hat{\mathbf{x}}^-$

Re-linearize at every time step — this is the EKF loop

Extending the State for GPS

To use pseudoranges directly we add the **receiver clock bias** b as a state. For our **2D example** (one new state on top of Block 5):

$$\mathbf{x} = \begin{bmatrix} p_x & p_y & v_x & v_y & b \end{bmatrix}^T \quad \text{(5 states)}$$

- Clock bias b [m] is the range-equivalent receiver timing error
- Modeled as slowly varying (random walk or Gauss-Markov)
- Every pseudorange equation includes b — the EKF estimates it automatically

Real-world 3D: add p_z, $v_z \Rightarrow$ 7-state vector $[p_x, p_y, p_z, v_x, v_y, v_z, b]^T$ — same structure

Adding one state unlocks full tight-coupling GPS integration

The Pseudorange Jacobian

Measurement function for satellite i:

$$h_i(\mathbf{x}) = \|\mathbf{p} - \mathbf{s}^{(i)}\| + b = \rho_i + b$$

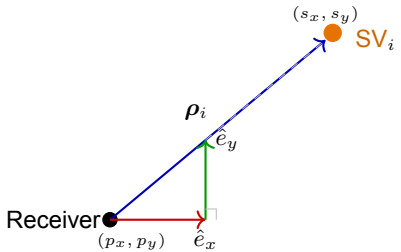
Partial derivatives give the row $\mathbf{H}_i = \dfrac{\partial h_i}{\partial \mathbf{x}}$.

2D example (5-state $[p_x, p_y, v_x, v_y, b]^T$):

$$\mathbf{H}_i = \Big[\underbrace{\hat{e}_x,\ \hat{e}_y}_{\text{unit vec.}},\ 0,\ 0,\ 1\Big]_{1\times 5}$$

where $\hat{e}_x = \dfrac{p_x - s_x^{(i)}}{\rho_i}, \quad \hat{e}_y = \dfrac{p_y - s_y^{(i)}}{\rho_i}$

3D extends to $[\hat{e}_x, \hat{e}_y, \hat{e}_z, 0, 0, 0, 1]_{1\times 7}$



$\mathbf{H}_i$ points from receiver to satellite – geometry matters!

EKF Predict-and-Update Loop

Propagate (same as linear KF):

$$\hat{\mathbf{x}}^- = \mathbf{F}\hat{\mathbf{x}}^+ \qquad \mathbf{P}^- = \mathbf{F}\mathbf{P}^+\mathbf{F}^T + \mathbf{Q}$$

Update — one satellite at a time:

$$r_i = \tilde{\rho}_i - h_i(\hat{\mathbf{x}}^-) \qquad \text{(nonlinear innovation)}$$

$$\mathbf{K}_i = \mathbf{P}^-\mathbf{H}_i^T\left(\mathbf{H}_i\mathbf{P}^-\mathbf{H}_i^T + \sigma_\rho^2\right)^{-1}$$

$$\hat{\mathbf{x}}^+ = \hat{\mathbf{x}}^- + \mathbf{K}_i\, r_i \qquad \mathbf{P}^+ = \mathbf{P}^- - \mathbf{K}_i\mathbf{H}_i\mathbf{P}^-$$

Repeat the update step for each satellite in view

Quick Exercise: Full EKF Update Step

Given (5-state: $[p_x,\ p_y,\ v_x,\ v_y,\ b]^T$):

$$\hat{\mathbf{x}}^- = \begin{bmatrix} 0 \\ 0 \\ 10 \\ -5 \\ 20 \end{bmatrix} \text{m}, \quad \mathbf{P}^- = 100\,\mathbf{I}_5, \quad \mathbf{s} = \begin{bmatrix} 3000 \\ 4000 \end{bmatrix} \text{m}, \quad \tilde{\rho} = 5035 \text{ m}, \quad \sigma_\rho^2 = 100 \text{ m}^2$$

- **Linearize:** Compute $\rho = \|\hat{\mathbf{p}}^- - \mathbf{s}\|$, predicted pseudorange $h = \rho + b$, and Jacobian $\mathbf{H} = [\hat{e}_x,\ \hat{e}_y,\ 0,\ 0,\ 1]$
- **Innovate:** Compute $r = \tilde{\rho} - h(\hat{\mathbf{x}}^-)$
- **Gain:** Compute $\mathbf{K} = \mathbf{P}^-\mathbf{H}^T\left(\mathbf{H}\mathbf{P}^-\mathbf{H}^T + \sigma_\rho^2\right)^{-1}$
- **Update:** Compute $\hat{\mathbf{x}}^+ = \hat{\mathbf{x}}^- + \mathbf{K}\,r$. Which states changed? Why?

Quick Exercise: Full EKF Update Step (Workspace)

Given: $\hat{\mathbf{x}}^- = [0,\ 0,\ 10,\ -5,\ 20]^T$ m, $\mathbf{P}^- = \text{diag}(100)$, $\mathbf{s} = [3000,\ 4000]^T$ m, $\tilde{\rho} = 5035$ m, $\sigma_\rho^2 = 100$ m^2

Demo: EKF with GPS Pseudoranges

Scenario: Vehicle navigates in 2D; GPS pseudoranges update an EKF

State model (5 states)

- 2D position, 2D velocity
- Receiver clock bias

Key observations

- EKF relinearizes **H** at every step as state estimate changes
- One sequential update per satellite

Demo: Extended Kalman Filter

MATLAB version in

`code/ExtendedKalmanFilter5D.m`

Wrap-Up

- **Loose coupling:** GPS solver outputs position; linear $\mathbf{H} = [\mathbf{I} \mid \mathbf{0}]$ — standard KF applies
- **Tight coupling:** pseudoranges fed directly; nonlinear $h_i(\mathbf{x})$ requires linearization $\rightarrow$ EKF
- The **pseudorange Jacobian** $\mathbf{H}_i = [\hat{e}_x,\ \hat{e}_y,\ \ldots,\ 1]$ encodes satellite geometry
- EKF **relinearizes** at each step: innovation uses $h(\hat{\mathbf{x}}^-)$, Kalman gain uses $\mathbf{H}|_{\hat{\mathbf{x}}^-}$
- Adding clock bias b to the state closes the loop — no separate GPS solver needed

Next block: the EKF assumes well-behaved sensors. What happens when one lies? $\rightarrow$ **fault detection and integrity**.

The EKF is optimal fusion applied to nonlinear sensors

SY6301 – Navigation and State Estimation

Block 8: Fault Detection, Integrity, and HMI

Block Objectives

By the end of this block you should be able to:

- Define a **sensor fault** and explain why it causes the Kalman filter innovation to grow
- Apply the **Mahalanobis distance test** to determine when an innovation is statistically suspicious
- Distinguish between **filter error bounds** (from **P**) and **integrity bounds** (protection levels)
- Define **Hazardous Misleading Information (HMI)** and identify it from position error and protection level time histories

What Is a Sensor Fault?

A fault is any condition where the measurement no longer behaves according to the assumed model:

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

Common fault sources in GPS/navigation:

- **Spoofing / jamming** — intentional range bias injected by an adversary
- **Unmodeled bias** — satellite clock anomaly, thermal drift, multipath
- **Wrong model parameters** — incorrect lever arm, timing offset, scale factor
- **Geometry collapse** — satellite dropout, poor PDOP

Key point: a fault is not just “bad noise” — it *violates the distributional assumptions* of the estimator, so the filter ingests it silently and corrupts its state estimate.

The Innovation Under a Fault

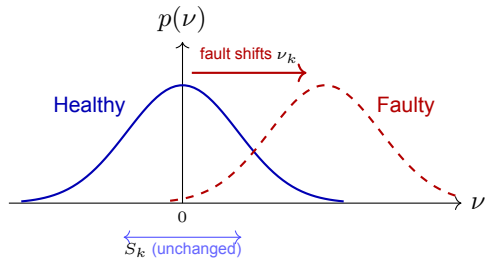
Under **healthy** conditions, the innovation is zero-mean:

$$\boldsymbol{\nu}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_k^-) \sim \mathcal{N}(\mathbf{0}, \mathbf{S}_k)$$

Under a **ramp fault** on one pseudorange, $z_k = h(x_k) + v_k + b_f(t_k)$, so the innovation becomes:

$$\nu_k \approx \underbrace{v_k}_{\text{noise}} + \underbrace{b_f(t_k)}_{\substack{\text{growing} \\ \text{bias}}}$$

- $S_k = \mathbf{H}\mathbf{P}^-\mathbf{H}^T + \sigma_\rho^2$ **does not change**
- ν_k grows steadily while S_k stays small



Result: Filter is confidently wrong.

How Big Is Too Big? — Mahalanobis Distance

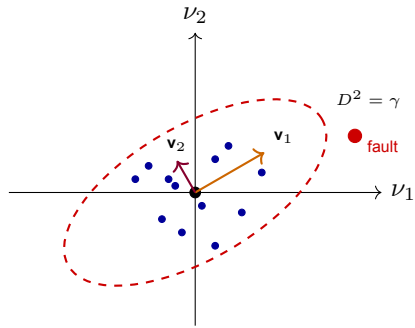
In **1D**, checking if a residual is suspicious is simple:

$$\frac{|\nu|}{\sigma} = \text{number of “sigmas”}$$

With **vector measurements** we need a generalization that accounts for covariance **S**.

Mahalanobis distance:

$$D^2 = \boldsymbol{\nu}^T \mathbf{S}^{-1} \boldsymbol{\nu}$$



D^2 measures how statistically surprising the innovation is, regardless of dimensions.

- Weights each residual component by its uncertainty
- Accounts for correlations between components

Detection as a Hypothesis Test

Under healthy conditions, D^2 follows a chi-squared distribution:

$$D^2 = \boldsymbol{\nu}^T \mathbf{S}^{-1} \boldsymbol{\nu} \;\sim\; \chi^2(m)$$

where m is the number of measurements stacked in $\boldsymbol{\nu}$.

Decision rule:

$$D^2 \underset{\text{healthy}}{\overset{\text{fault}}{\gtrless}} \gamma, \qquad \gamma = \chi^2_{m,\,1-P_{FA}}$$

Threshold γ is chosen from the desired **false alarm probability** P_{FA}:

Measurements m		$P_{FA} = 1\%$	$P_{FA} = 0.3\%$
1	(single pseudorange)	6.63	8.83
4	(batch, 4 satellites)	13.28	16.25

Quick Exercise: Is This Residual Suspicious?

Scenario: Satellite 1 has a ramp fault with rate $\dot{b}_f = 3$ m/s injected at $t_0 = 10$ s. The innovation variance is $S = 5^2 = 25$ m^2 (from $S = \mathbf{H}\mathbf{P}^{-}\mathbf{H}^T + \sigma_\rho^2$).

- **Detection:** At $t = 15$ s the pseudorange innovation is $\nu = 15$ m. Compute $D^2 = \nu^2/S$. Is this suspicious at $P_{FA} = 0.3\%$ ($m = 1$, $\gamma = 8.83$)?
- **Time-to-detect:** The bias grows linearly from zero. At what time does D^2 first exceed $\gamma = 8.83$? What is the time-to-detect $T_D = t_D - t_0$?
- **Trade-off:** If you tighten the threshold to $\gamma = 4.0$ (more sensitive), what happens to T_D? What is the cost?

Quick Exercise: Is This Residual Suspicious? (Workspace)

Given: $\dot{b}_f = 3$ m/s, $t_0 = 10$ s, $\sigma_S = 5$ m, $\gamma = 8.83$ ($m = 1$, $P_{FA} = 0.3\%$)

What Happens After Detection?

Detection alone is not enough — the system must **respond**.

Common fault responses:

- **Fault exclusion** — remove the offending satellite; continue with remaining SVs
- **Fault accommodation** — inflate **R** for the suspect SV, or augment state with a bias term
- **Fault recovery** — reintroduce the sensor once its innovations return to nominal
- **Multi-filter / parallel hypothesis** — run N sub-filters each excluding one SV; the sub-filter that remains statistically consistent identifies the culprit

The F-47 ANS uses a multi-filter architecture: one main filter plus sub-filters each excluding one sensor. This produces a **position integrity output** that is separate from the filter covariance **P**.

Navigation Integrity vs. Accuracy

These answer fundamentally different questions:

Accuracy	How close is my estimate to the truth?
Integrity	Can I *trust* this estimate right now?

Operationally, integrity is a system's ability to:

- Detect when its performance is unacceptable
- Alert the operator within a specified time
- Bound the risk of an **undetected failure**

Integrity is not about average performance;
it is a guarantee about worst-case behavior.

Error Bounds vs. Integrity Bounds — Part 1

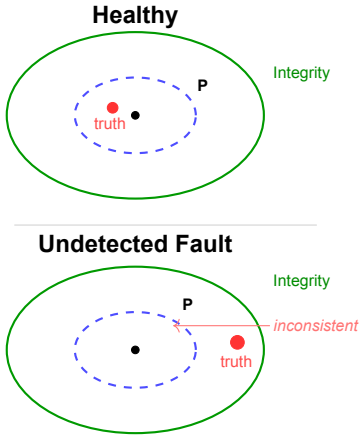
Filter Error Bound (from P)

- Derived from filter covariance: $\mathbf{P}_{pos} \Rightarrow$ error ellipse
- Interpretation: *"If the model is correct, true error lies here with high probability"*
- An undetected fault corrupts $\hat{\mathbf{x}}$ but does *not* grow **P**

Inconsistency:

$$\text{Actual error} \not\sim \mathcal{N}(\mathbf{0}, \mathbf{P})$$

The filter stays "confidently wrong" — the **P** ellipse no longer contains truth.



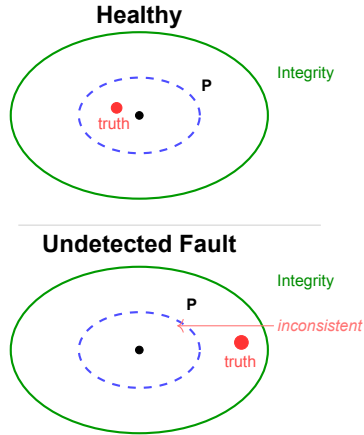
Error Bounds vs. Integrity Bounds — Part 2

Integrity Bound

- A separate, more conservative bound from the multi-filter architecture
- **Remains valid even when one sensor is faulty**
- Expressed as spatial thresholds: HPL and VPL

Healthy system: truth inside both **P** and integrity bound

Faulty system: truth outside **P** (inconsistent), but *still inside* integrity bound — no HMI yet



Protection Levels

The integrity bound is expressed as usable spatial thresholds — the **protection levels**.

Horizontal Protection Level (HPL):

$$\text{HPL} = n_\alpha \sqrt{P_{\text{int},N} + P_{\text{int},E}}$$

Vertical Protection Level (VPL):

$$\text{VPL} = n_\alpha \sqrt{P_{\text{int},D}}$$

- P_{int} = diagonal elements of the **integrity covariance** (not the filter covariance **P**)
- n_α = multiplier for desired containment probability (e.g., $n_\alpha = 2.576$ for 99%)
- HPL/VPL are **not** confidence ellipses from the EKF — they are guaranteed performance zones from the multi-filter architecture, valid under a single sensor fault

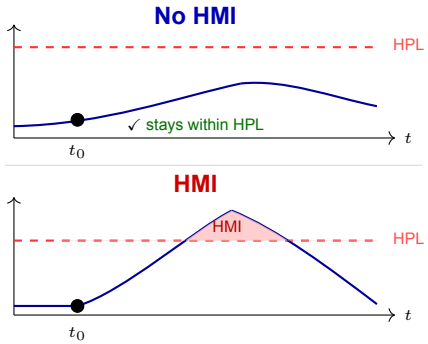
Hazardous Misleading Information (HMI)

Definition: HMI occurs when the **true position error exceeds the protection level** while the fault is undetected:

$\text{HMI}_H: \ e_H > \text{HPL} \qquad \text{HMI}_V: \ |e_D| > \text{VPL}$

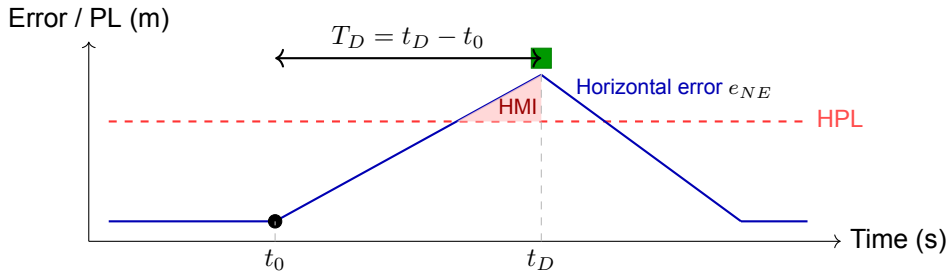
- **Safe outcome:** error grows but stays within HPL (integrity bound covers the fault scenario)
- **HMI outcome:** error exceeds HPL before or during the detection window — system actively misleads the operator

HMI is the most critical failure mode in nav systems.



Reading the Plots: Measuring TTD and HMI Exposure

As tester, you observe position error and protection level over time:



HMI exposure = total time in $[t_0, t_D]$ where $e_{NE} >$ HPL (or $|e_D| >$ VPL)

Demo: EKF with an Injected Fault

Using EKF demo from last time:

- Ramp bias injected on satellite 1 at `t_fault`
- Compute D_k^2 per satellite update; compare to threshold
- Plot sat innovations — one grows, others stay small
- Plot position error vs. HPL; shade HMI exposure window

Things to observe:

- How does D^2 change before vs. after the fault starts?
- How does pos error compare to the reported **P** ellipse?

Demo: Fault Detection

MATLAB version in `code/FaultDetectionDemo.m`

Wrap-Up

- A **sensor fault** violates the measurement model — the filter ingests it silently and corrupts its state estimate
- The **innovation** $\boldsymbol{\nu}_k$ carries a growing bias under a fault; the **Mahalanobis distance** $D^2 = \boldsymbol{\nu}^T \mathbf{S}^{-1} \boldsymbol{\nu}$ measures how statistically surprising it is
- Fault detection is a **hypothesis test:** $D^2 \gtrless \gamma$, where γ trades false alarm rate against detection sensitivity
- **Integrity** $\neq$ **accuracy:** the filter covariance **P** becomes misleading under a fault; the **integrity bound** (HPL/VPL) remains valid
- **HMI** occurs when the position error exceeds the protection level — the most critical navigation failure mode

Next block: apply everything to a real test plan $\rightarrow$ **F-47 ANS Project**.

SY6301 – Navigation and State Estimation

Block 9: Project Introduction

Block Objectives

By the end of this block you should be able to:

- Explain why correlated time-series data requires **effective sample size** N_{eff}, not raw sample count N
- Estimate T_{corr} from an autocorrelation function and explain the **10% rule of thumb**
- Derive N_{eff} from first principles and **plan a statistically defensible test duration**
- Apply the **95th-percentile ECDF** for accuracy requirements
- Estimate inertial **drift rate** with a confidence interval on the slope
- Infer **fault detection time** from navigation error behavior without FDE flags

F-47 ANS — Background and Mission Need

The F-47 Navigation Computer has been upgraded with an **Alternative Navigation Suite (ANS)** to sustain performance in **GPS-denied environments**.

The ANS fuses inertial measurements with:

- GPS (when available)
- AltNav aiding: MagNav, VisNav, SoPNav

When GPS is denied, the ANS transitions to AltNav-aided operation while continuing to provide **integrity-relevant uncertainty outputs**.

F-47 ANS — Program Status

HITL testing is complete. Key findings:

- Horizontal navigation error has $T_{\text{corr}} \approx 15$ s during steady-state operation
  $\Rightarrow$ statistical foundation for planning effective independent sample size
- Over 200 fault events executed across step bias, ramp bias, and others
- **Most stressing case:** single-SV ramp bias—longest T_D, greatest HMI exposure
- DT will focus on SV ramp bias to bound worst-case fault detection performance

Critical DT constraint:

- SPO directed production-representative sensors and avionics bus data only
- No dedicated Flight Test Instrumentation–no internal fault flags
  $\Rightarrow$ detection time t_D must be *inferred* from navigation solution behavior

System Under Test — ANS Modes and Outputs

ANS Operating Modes

Mode	Sensor Suite
AllSource	Inertial + GPS + AltNav
AltNav	Inertial + AltNav (GPS denied)
Inertial	Inertial only

Truth position (LLH) collected via an independent truth reference system.

Outputs Available

- True position (LLH)
- Estimated position (LLH)
- Estimated position covariance (N-E-D)
- Integrity protection levels: HPL, VPL

These outputs form the basis of **all four compliance assessments**.

System Under Test — Datasets Provided

File	Mode	Purpose
`data_AllSource`	AllSource	Accuracy assessment
`data_AltNav`	AltNav	Accuracy assessment
`data_Inertial`	Inertial	Drift assessment
`data_Spoof`	AllSource	Fault detection / integrity

Each dataset contains: time (s), true position (LLH), estimated position (LLH), covariance (N-E-D), integrity (HPL/VPL), and fault onset times t_0 (Spoof only).

Critical warning: `data_Spoof` must **not** be used for accuracy assessment. Fault injection events produce episodically elevated errors that inflate the 95th-percentile estimate. **Accuracy and fault-detection testing require separate dedicated runs.**

F-47 ANS Performance Requirements

Req	Mode	Threshold	Sample Requirement
1	AllSource	$H_{95} \leq 1.0$ m, $V_{95} \leq 5.0$ m	$N_{\text{eff}} \geq 300$
2	AltNav	$H_{95} \leq 5.0$ m, $V_{95} \leq 10.0$ m	$N_{\text{eff}} \geq 300$
3	Inertial	Drift rate ≤ 1.0 NM/hr (upper 95% CI)	$T \geq 85$ min
4	AllSource	$T_D \leq 5$ s, $\text{HMI}_H \leq 1$ s, $\text{HMI}_V \leq 1$ s	27 of 30 events

Reqs 1 & 2 require $N_{\text{eff}} \geq 300$ effective independent samples — the correlation structure of navigation error drives the minimum run duration.

Reqs 3 & 4 specify the statistical method directly: upper CI bound and 27/30 rule.

The Problem with Correlated Data

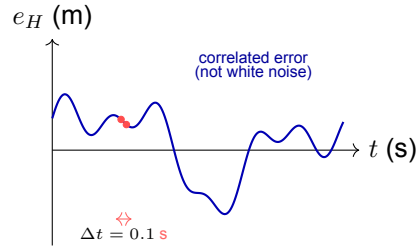
A 15-minute AllSource run at 10 Hz yields $N = 9{,}000$ samples.

Tempting conclusion:

$$\text{CI width} \propto \frac{1}{\sqrt{9{,}000}} \approx 0.011 \quad \text{(very tight!)}$$

The problem:

- Consecutive error samples are *not* independent
- $e_{k=1}$ and $e_{k=2}$ are essentially the same number
- Using $N = 9{,}000$ falsely assumes independence



Actual effective sample count is much smaller than $N = 9{,}000$.

The Autocorrelation Function

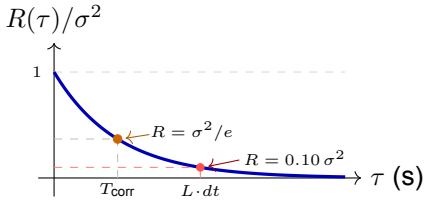
The **ACF** measures how similar a signal is to a time-shifted copy:

$$R(\tau) = \mathbb{E}\big[e(t)\, e(t+\tau)\big]$$

For navigation error driven by an AR(1) process:

$$\boxed{R(\tau) = \sigma^2\, e^{-|\tau|/T_{\text{corr}}}}$$

- T_{corr}: Correlation time — the "memory" of the error
- At $\tau = 0$: $R(0) = \sigma^2$ (variance)
- At $\tau = T_{\text{corr}}$: $R = \sigma^2/e \approx 0.37\,\sigma^2$



HITL result (given): $T_{\text{corr}} \approx 15\,\text{s}$

The 10% Rule of Thumb — How Close Is It?

Standard formula (from variance-of-the-mean derivation):

$$N_{\text{eff}} = \frac{T_{\text{total}}}{2\,T_{\text{corr}}}$$

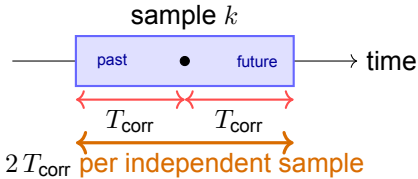
Method	Implicit ρ threshold	Difference
Standard (sum of ACF)	$\rho = e^{-2} \approx 13.5\%$	—
10% decimation rule	$\rho = 10.0\%$	-13%

The 10% rule gives an N_{eff} about 13% smaller relative to the standard formula.

Effective Sample Size — Where Does the Factor of 2 Come From?

The factor of 2 appears because the ACF sum is prefixed by 2 in the variance formula.

Physical interpretation: each sample is correlated with neighbors both **forward and backward** in time. The two-sided correlation window spans $2\,T_{\text{corr}}$ of the timeline.



Each “independent look” therefore occupies $2\,T_{\text{corr}}$ seconds:

$$N_{\text{eff}} = \frac{T_{\text{total}}}{2\,T_{\text{corr}}}$$

Test Duration Planning

Given: $T_{\text{corr}} = 15\,\text{s}$, $N_{\text{eff}} \geq 300$ required for accuracy assessments.

Solve for minimum run duration:

$$N_{\text{eff}} = \frac{T_{\text{total}}}{2\,T_{\text{corr}}} \geq 300 \quad \Longrightarrow \quad T_{\text{total}} \geq 2 \times 15 \times 300 = 9{,}000\,\text{s} \approx 150\,\text{min}$$

Run duration	N (at 10 Hz)	N_{eff}	Meets $N_{\text{eff}} \geq 300$?
15 min	9,000	30	**No**
60 min	36,000	120	**No**
150 min	90,000	300	**Yes**

95% Statistical Methods — Which Tool Matches the Requirement?

The F-47 ANS requirement reads:

"Horizontal position error shall be ≤ 1.0 *m (95% confidence)."*

- **95th-percentile ECDF**
  Find the value below which 95% of error samples fall:

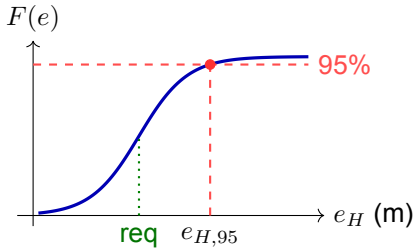
$$H_{95} = \texttt{prctile(errH, 95)}$$

- **95% CI on the mean**
  Tests whether the *mean* is below a bound. The requirement constrains the *tail*, not the mean.

Reading the ECDF

Steps to assess compliance:

- Compute horizontal errors e_H vs. truth
- Build the empirical CDF (`ecdf` in MATLAB)
- Read off $e_{H,95}$ — value where $F(e) = 0.95$
- Compare: **PASS** if $e_{H,95} \leq 1.0$ m



Inertial Drift — The Error Model

In inertial-only mode, velocity bias random walk **integrates** into position. Horizontal error grows approximately linearly:

$$e_H(t) = a + b\,t + \varepsilon(t)$$

Goal: estimate slope b (drift rate in m/s) and verify ≤ 1.0 NM/hr.

OLS estimate using MATLAB backslash:

$$\hat{\boldsymbol{\beta}} = \mathbf{X} \backslash \mathbf{e}_H, \qquad \mathbf{X} = \begin{bmatrix} \mathbf{1} & \mathbf{t} \end{bmatrix}$$

gives $\hat{\boldsymbol{\beta}} = [\hat{a},\ \hat{b}]^\top$ — intercept and slope.

Unit conversion:

$$\hat{b}\,[\text{m/s}] \times \frac{3600}{1852} = \hat{b}\,[\text{NM/hr}]$$

Inertial Drift — 95% CI and Pass Criterion

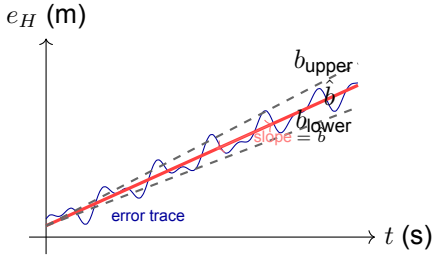
95% CI on slope $\hat{b}$:

$$\hat{b} \pm t_{N-2,\,0.975} \cdot \hat{\sigma}_b$$

Pass criterion:

$$b_{\text{upper}}\,[\text{NM/hr}] \leq 1.0$$

Compare the **upper bound**, not the point estimate. Even if $\hat{b}$ is below 1.0 NM/hr, a large uncertainty could still result in FAIL.



Fault Detection Without FTI — The Algorithm

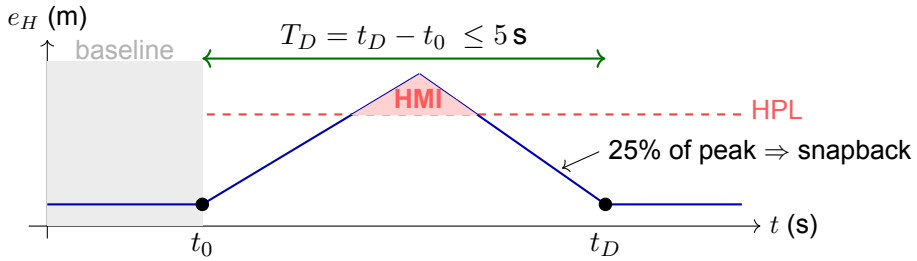
Constraint: production sensors only — no FTI, no internal fault flags. Fault onset time t_0 is known from the test log. Detection time t_D must be *inferred* from navigation error behavior.

Peak-and-snapback algorithm:

- Compute **pre-fault baseline** from error in window $[t_0 - T_{\text{pre}},\ t_0)$
- Find **peak error** e_{peak} in the post-fault search window
- Set **snapback threshold**: $e_{\text{thr}} = e_{\text{base}} + \alpha\,(e_{\text{peak}} - e_{\text{base}}), \quad \alpha = 0.25$
- t_D = first time after peak where $e_H \leq e_{\text{thr}}$ for ≥ 0.5 s

Once t_D is identified, compute HMI exposure during $[t_0, t_D]$ and check $T_D = t_D - t_0$.

Fault Detection Without FTI — Timeline and HMI



27/30 Compliance Rule

Requirement 4 states: for 30 independent spoof events, at least 27 of 30 shall satisfy:

Metric	Threshold
Time-to-detect $T_D = t_D - t_0$	$\leq 5\,\text{s}$
Horizontal HMI exposure during $[t_0,\, t_D]$	$\leq 1\,\text{s}$
Vertical HMI exposure during $[t_0,\, t_D]$	$\leq 1\,\text{s}$

The 27/30 threshold was established by the SPO to reflect a probabilistic performance specification. The system is not required to be perfect on every event, but must demonstrate reliable, consistent performance across the test campaign.

MATLAB Script Map — Five Focused Parts

Each script is self-contained. Work through them in order.

Script	Task	Req
`Project_DataRequest.m`	Compute N_{eff}, justify run durations	Planning
`Project_CorrelationTime.m`	ACF from data, estimate T_{corr}, verify N_{eff}	Planning
`Project_AccuracyECDF.m`	95th-pct ECDF, compare to bounds	1 & 2
`Project_InertialDrift.m`	OLS slope, 95% CI, NM/hr comparison	3
`Project_FaultDetection.m`	Infer t_D, HMI exposure, 27/30 count	4

Wrap-Up

- Correlated time series have far fewer **independent samples** than the raw count N suggests — accuracy assessments require $N_{\text{eff}} = T_{\text{total}}/(2\,T_{\text{corr}})$
- T_{corr} is read from the ACF; the **10% rule** provides a quick estimate within $\sim 13\%$ of the standard formula
- Accuracy is a **tail** property — assess with the **95th-percentile ECDF**, not a CI on the mean
- Inertial drift is the OLS **slope** of $e_H(t)$; compliance compares the **upper 95% CI bound** to the requirement
- Without FTI, t_D is inferred from the **peak-and-snapback** behavior of the navigation error
- The **27/30 rule** converts a probabilistic specification into a finite-sample pass/fail criterion

Statistical planning is the difference between data and evidence.

UNITED STATES AIR FORCE
TEST PILOT SCHOOL

TESTERS
LEADERS
THINKERS
INNOVATORS

FLIGHT TEST COURSE • SPACE TEST COURSE • ENLISTED TEST COURSE