An Introduction to the Kalman Filter

A Recursive Algorithm for State Estimation

Emre Girgin

Space Robotics and Generative Estimation

July 10, 2025

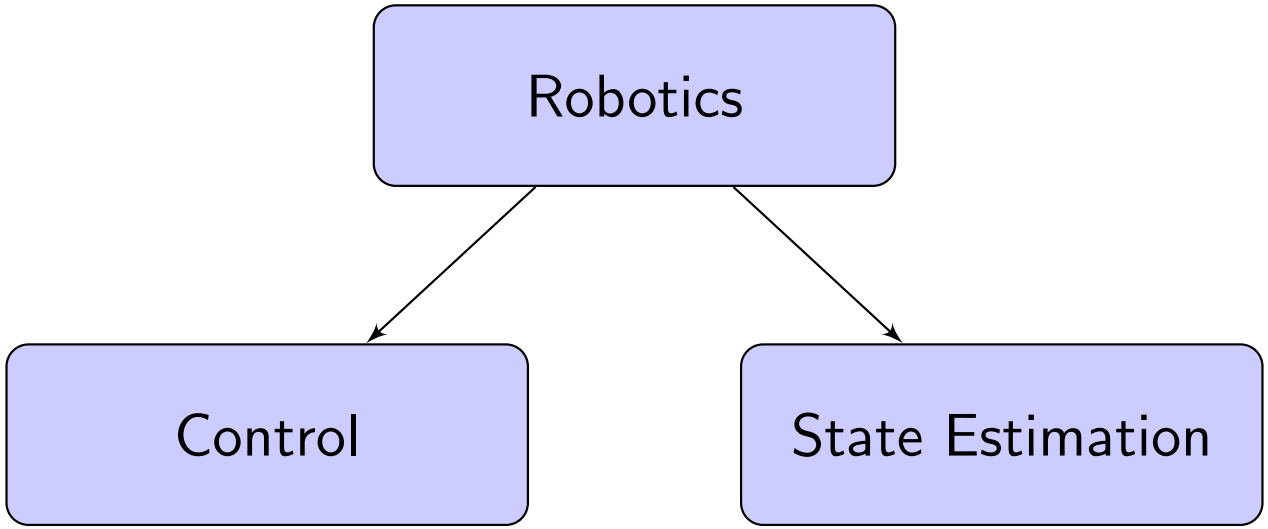
Primary Reference and Disclaimer

The theoretical content of this presentation is largely based on the book **Optimal State Estimation: Kalman, H-infinity, and Nonlinear Approaches** by *Dan Simon*.
A complete list of references is available at the end.

Disclaimer

The slides and accompanying code are for educational purposes and may contain errors. Please verify any critical information before use.

Robotics, Control, and State Estimation



And, although specific robots have their subtleties, there are also some common issues we must face in all applications, particularly **state estimation** *and* **control**.

from *State Estimation for Robotics*, Timothy D. Barfoot.

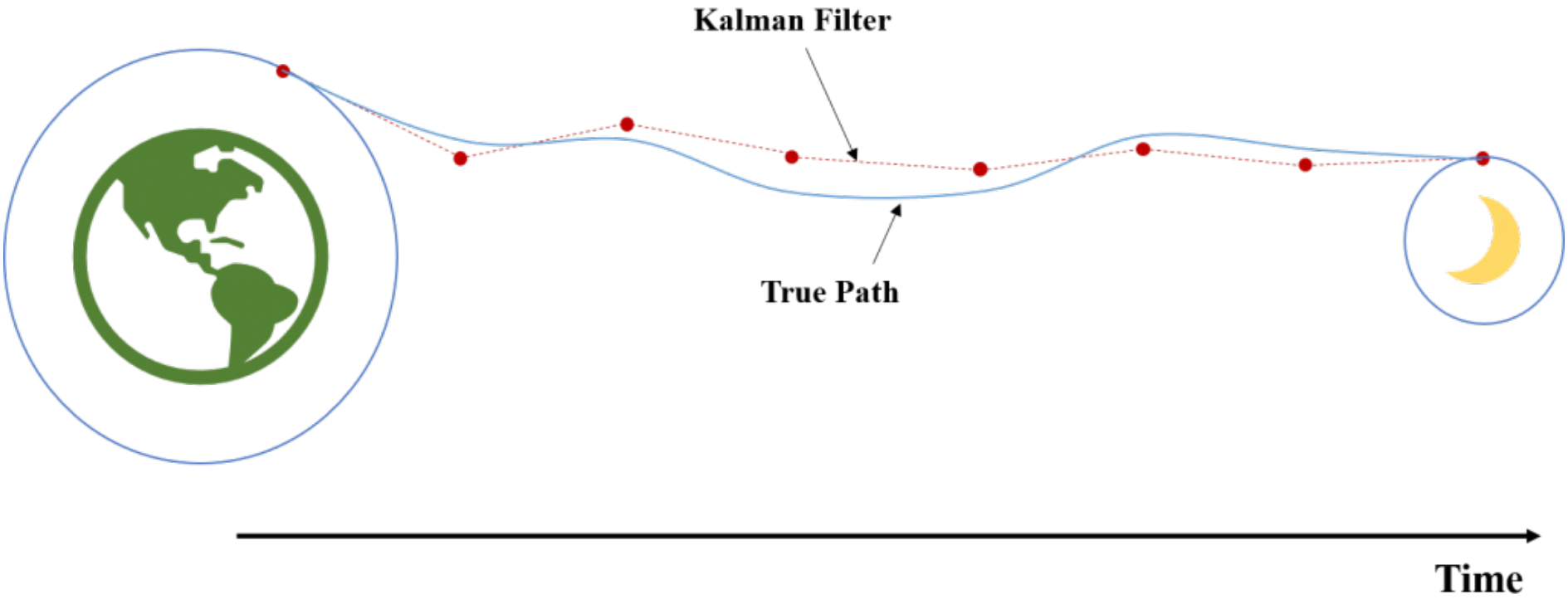
- **Control**:The process of commanding a robot's motors and actuators to make it achieve a specific goal or follow a desired trajectory.
- **State Estimation**: The process of using sensor data to determine the robot's current configuration, such as its position, orientation, and velocity.

Overview and Motivation

- **The Goal:** Kalman Filters are designed to estimate unknown variables from noisy data. This makes them ideal for solving *state estimation* problems.
- **The Analogy:** Think of *state estimation* as the general problem, like needing to build something. The Kalman Filter is a specific, powerful **tool** for that job, like a high-precision screwdriver.
- **The Impact:** For decades, it has been an industry standard, crucial for applications in aerospace, robotics, and finance. Its relevance in both academia and industry remains strong today.

Key Applications

- **Navigation:** GPS, inertial navigation systems (INS).
- **Robotics:** Sensor fusion for localization and mapping (SLAM).
- **Finance:** Time series analysis and prediction.
- **Computer Vision:** Object tracking.



Assumed Knowledge

A basic familiarity with the following concepts will be very helpful:

- **Linear Algebra:** The Kalman Filter is built on vectors and matrices. We'll be using matrix operations to represent the state of a system and its transformations.
- **Calculus:** We will take derivatives. (Mostly only Jacobians)
- **Probability & Statistics:** Real-world data is noisy. We use concepts like mean and variance to model this uncertainty, treating our estimates as probability distributions.
- **Application Context (Robotics):** Our examples will be drawn from robotics, such as vehicle navigation. No expertise is required, but the context helps illustrate the filter's power.

Theoretical Perspectives

The Kalman Filter's equations can be understood from several complementary viewpoints:

- **The Optimization View:** Finding the state estimate that minimizes prediction error. (*Today's presentation!*)
- **The Probabilistic View:** Finding the most likely state given the measurements and our prior knowledge. (*Recently become more popular!*)
- **The Control Theory View:** Using a system model to predict the state and measurements to provide corrective feedback. (AE 527)

Outline

Section 1.1: Least Square Solution

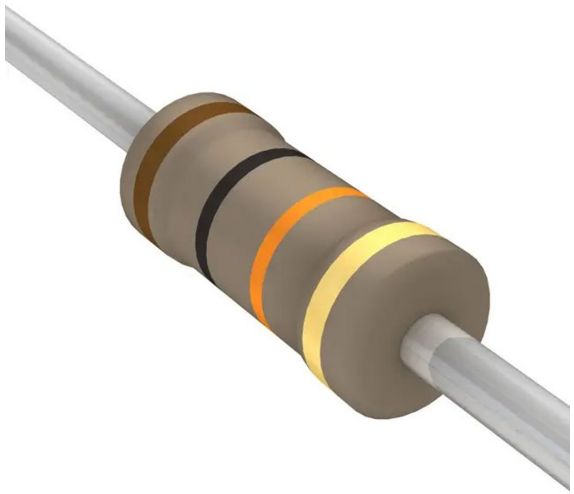
Estimating a Constant

Suppose we have a resistor but we do not know its resistance.

Objective: Estimate the unknown, resistance (R).

Method: Measure using a noisy multimeter.

Challenge: Each measurement $y = R + v$ includes random noise (v).



The Measurement Model

Our goal: find the best estimate $\hat{\mathbf{x}}$, using a set of measurements $\mathbf{y}$.

We assume a linear relationship:

Linear Measurement Model

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v}$$

- $\mathbf{y}$ is the vector of our noisy measurements.
- $\mathbf{H}$ is the measurement matrix.
- $\mathbf{x}$ is the true, unknown vector we want to estimate.
- $\mathbf{v}$ is the vector of unknown measurement noise.

The Least-Squares Approach

The "best" estimate (formalized by Gauss) is the one that **minimizes the error** between our measurements $\mathbf{y}$ and the predictions $\mathbf{H\hat{x}}$.

This error is called the **measurement residual**, $\epsilon_{\mathbf{y}}$.

Cost Function (J)

We minimize the sum of the squared residuals, defined by the cost function J:

$$J = {\epsilon_{\mathbf{y}}}^T \epsilon_{\mathbf{y}} = (\mathbf{y} - \mathbf{H\hat{x}})^T (\mathbf{y} - \mathbf{H\hat{x}})$$

The $\hat{\mathbf{x}}$ that minimizes J is our optimal estimate.

Question?

Why do we minimize the sum of the squares of the error, and not just the sum of the error? $\hat{J} = \epsilon_{\mathbf{y}} = (\mathbf{y} - \mathbf{H\hat{x}})$?

Why Minimize the Sum of Squares?

Question?

Why do we minimize the sum of the squares of the error, and not just the sum of the error? $\hat{J} = \epsilon_{\mathbf{y}} = (\mathbf{y} - \mathbf{H}\hat{\mathbf{x}})$?

Answer

- **To prevent cancellation:** Positive error (+10) and negative error (-10) would cancel out to zero, falsely suggesting a perfect fit. So why not absolute value of the error ?
- **To penalize large errors:** Squaring makes large errors count for much more than small ones (e.g., $10^2 = 100$ vs $2^2 = 4$). This forces the solution to avoid large deviations.
- **For a unique solution:** A squared error function is convex (it has a single minimum), which guarantees we can find the one best answer using calculus.

The Optimal Estimate

To minimize J, we take its partial derivative with respect to $\hat{\mathbf{x}}$ and set it to zero.

Minimization Condition

$$\frac{\partial J}{\partial \hat{\mathbf{x}}} = -2\mathbf{H}^T\mathbf{y} + 2\mathbf{H}^T\mathbf{H}\hat{\mathbf{x}} = 0$$

Solving for $\hat{\mathbf{x}}$ gives us the celebrated **least-squares solution**:

The Solution (Normal Equation)

$$\hat{\mathbf{x}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$$

This provides the best estimate of $\mathbf{x}$ in the least-squares sense.

Question?

From $\mathbf{y} = \mathbf{H}\mathbf{x}$, why can't we just solve for $\mathbf{x}$ by $\mathbf{H}^{-1}$?

$$\mathbf{x} = \mathbf{H}^{-1}\mathbf{y} \quad \text{Why not this?}$$

A Question on Inverses

Question?

From $\mathbf{y} = \mathbf{H}\mathbf{x}$, why can't we just solve for $\mathbf{x}$ by $\mathbf{H}^{-1}$?

$$\mathbf{x} = \mathbf{H}^{-1}\mathbf{y} \quad \text{Why not this?}$$

Answer

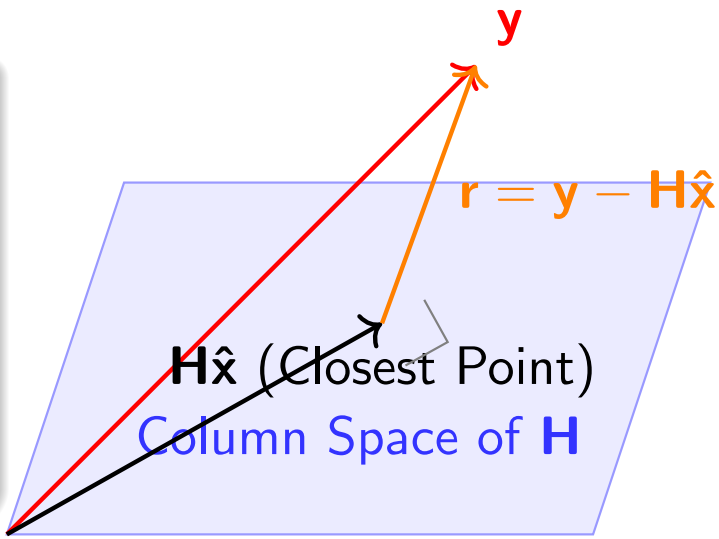
Because $\mathbf{H}$ is usually **not invertible**!

- In most real-world problems, we have many more measurements than state variables (e.g., 100 measurements for 2 states).
- This means $\mathbf{H}$ is a "tall," non-square matrix. Only square matrices can have a regular inverse.
- The term $(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$ is the next best thing: it's called the **pseudoinverse** (or left inverse). It gives us the best solution even when a true inverse doesn't exist.

Why "Normal"?: Geometric Interpretation

The Core Problem

Usually, our measurement vector $\mathbf{y}$ does not lie in the subspace spanned by the columns of $\mathbf{H}$ (the "Column Space").
This means there is **no exact solution** to $\mathbf{Hx} = \mathbf{y}$.



The Least Squares Goal

Find the vector $\hat{\mathbf{x}}$ such that the projection $\mathbf{H}\hat{\mathbf{x}}$ is the **closest possible point** within the Column Space to our actual measurement vector $\mathbf{y}$. The shortest path from a point to a plane is always **perpendicular**, or **normal**, to the plane.

Conclusion: The error (residual) vector $\mathbf{r}$ must be **normal** to the column space of $\mathbf{H}$.

Least-Squares in Action: The Resistor Example

Let's apply our general solution to the problem of estimating a resistor's constant value, x, from k noisy measurements.

Recall the Model:

$$\underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}}_{\mathbf{H}} x + \underbrace{\begin{bmatrix} v_1 \\ \vdots \\ v_k \end{bmatrix}}_{\mathbf{v}}$$

Recall the Solution:

$$\hat{x} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$$

Why H is all 1 in this case ?

Because we directly measure it. No need to any calculation.

Least-Squares in Action: The Resistor Example

Substituting our specific H and y:

$$\hat{x} = \left(\begin{bmatrix} 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}$$

$$= (k)^{-1}(y_1 + y_2 + \cdots + y_k)$$

$$= \frac{1}{k} \sum_{i=1}^{k} y_i$$

The Intuitive Result

The optimal least-squares estimate for a constant is simply the average of all measurements!

The Batch Least Squares Estimator

This vanilla method often called "batch" least squares, finds the single best-fit state vector ($\hat{\mathbf{x}}$) that minimizes the sum of squared errors across an **entire collection (a "batch") of measurements**.

Stacking the System

We collect all m measurements and stack them into a single system:

$$\underbrace{\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_m \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \vdots \\ \mathbf{H}_m \end{bmatrix}}_{\mathbf{H}} \mathbf{x} + \underbrace{\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_m \end{bmatrix}}_{\mathbf{v}}$$

All Data Required: You must have every measurement ($\mathbf{y}_1 \dots \mathbf{y}_m$) available before you can compute the solution.

Congratulations!

You have just learned the fundamentals of the **Least-Squares Estimation**.

Key Takeaways

You can now:

- Model a real-world problem with linear equations ($\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v}$).
- Define a cost function to quantify estimation error.
- Derive the optimal solution that minimizes this error.

This method is the essential foundation for more advanced topics.

Section 1.2: The Idea of Weighted Least Squares

Imagine measurements aren't of the same quality.

- Some measurements come from high-quality, low-noise sensors.
- Others come from cheap, high-noise sensors.

The Question: How do we use all the data without letting the noisy measurements corrupt our estimate?

The Principle

Never throw away data! Instead, give more weight to the measurements you trust and less weight to the ones you don't.

Quantifying Measurement Noise

Statistical Assumptions:

- Noise is **zero-mean**: $E[v_i] = 0$.
- All noise sources are **independent**.
- Each measurement may have a **unique variance**: $E[v_i^2] = \sigma_i^2$.

The Measurement Covariance Matrix ($\mathbf{R}$)

$$\mathbf{R} = E[\mathbf{v}\mathbf{v}^T] = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_k^2 \end{bmatrix}$$

The **diagonal** holds the variance (noise level) of each measurement. The **off-diagonals** are zero because the noise is independent.

Weighted Least-Squares (WLS) Cost Function

We minimize a **weighted sum of squares**, where each error is scaled by the inverse of its noise variance.

The Weighted Cost Function (J)

$$J = \frac{\epsilon_{y_1}^2}{\sigma_1^2} + \frac{\epsilon_{y_2}^2}{\sigma_2^2} + \cdots + \frac{\epsilon_{y_k}^2}{\sigma_k^2} \text{ , where } \epsilon_{\mathbf{y_k}} = (\mathbf{y_k} - \mathbf{H_k}\hat{\mathbf{x}}_\mathbf{k})$$

In matrix form, this becomes:

$$J = (\mathbf{y} - \mathbf{H}\hat{\mathbf{x}})^T \mathbf{R}^{-1} (\mathbf{y} - \mathbf{H}\hat{\mathbf{x}})$$

This automatically down-weights noisy measurements (with large σ_i^2), reducing their influence on the final estimate.

The Weighted Least-Squares Solution

The WLS Solution

$$\frac{\partial J}{\partial \hat{\mathbf{x}}} = 0 \quad \Longrightarrow \quad \hat{\mathbf{x}} = (\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{R}^{-1} \mathbf{y}$$

Important Condition

The noise matrix $\mathbf{R}$ must be invertible. This means every measurement must have some noise (no σ_i^2 can be zero).

Revisit Assumptions for Real-World

- **Zero-Mean Noise (No Bias):**
  - **The Problem:** A systematic offset, or bias **b**, can make measurements consistently wrong.
  - **The Fix:** If the bias is known, subtract it: $\mathbf{y}_{corrected} = \mathbf{y}_{original} - \mathbf{b}$. If unknown, it must be estimated. (Wait for Kalman Filter)

- **Independent Noise (No Correlation):**
  - **The Problem:** This is often **violated**. For instance, vibrations on a drone can introduce correlated errors across all its sensors simultaneously.
  - **The Fix:** This is handled in advanced filters by defining the relationships between noise sources. (Wait for Kalman Filter)

- **All Measurements Have Noise:**
  - **The Problem:** For the math to work, the noise matrix **R** must be invertible.
  - **The Fix:** This is a safe assumption. Every real-world sensor has some random noise, so this condition is always met.

WLS Example: The Resistor Problem

We return to estimating a resistor's value, x, from k noisy measurements. This time, we assume each measurement has a unique noise variance.

Measurement Model

Each of our k measurements is given by:

$$y_i = x + v_i \quad \text{with noise variance} \quad E(v_i^2) = \sigma_i^2$$

Matrix Form:

$$\underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}}_{\mathbf{H}} x + \underbrace{\begin{bmatrix} v_1 \\ \vdots \\ v_k \end{bmatrix}}_{\mathbf{v}}$$

Noise Covariance: The matrix $\mathbf{R}$ is diagonal, as the noise sources are independent:

$$\mathbf{R} = \text{diag}(\sigma_1^2, \ldots, \sigma_k^2)$$

WLS Example: Applying the Formula

We start with the general Weighted Least-Squares solution:

$$\hat{x} = (\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{R}^{-1} \mathbf{y}$$

Substituting our H and R

Plugging in the matrices for our resistor problem gives:

$$\hat{x} = \left(\begin{bmatrix} 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_k^2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \right)^{-1} \times \left(\begin{bmatrix} 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_k^2 \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix} \right)$$

WLS Example: The Intuitive Result

After performing the matrix multiplication from the previous slide:

The Final Estimate

$$\hat{x} = \left(\sum_{i=1}^{k} 1/\sigma_i^2 \right)^{-1} \left(\sum_{i=1}^{k} y_i/\sigma_i^2 \right) = \frac{\sum y_i/\sigma_i^2}{\sum 1/\sigma_i^2}$$

Conclusion

The optimal estimate is a **weighted average** of the measurements.

- Each measurement y_i is weighted by $1/\sigma_i^2$, the inverse of its variance. This is a measure of our **confidence** in that measurement.
- If all noise variances are equal ($\sigma_1^2 = \sigma_2^2 = \dots$), reduces to our first result.

Pop Quiz 1: What does R represent?

Scenario

You are tracking an object using two different sensors:

- A radar measures the object's **range** in meters (m).
- A thermal camera measures its **temperature** in Celsius (C).

Your measurement vector is $\mathbf{y} = \begin{bmatrix} \text{range} \\ \text{temperature} \end{bmatrix}$.

Question

What are the physical meanings and units of the diagonal elements of your 2×2 matrix $\mathbf{R}$, specifically R_{11} and R_{22}?

Pop Quiz 1: What does R represent?

Scenario

You are tracking an object using two different sensors:

- A radar measures the object's **range** in meters (m).
- A thermal camera measures its **temperature** in Celsius (C).

Your measurement vector is $\mathbf{y} = \begin{bmatrix} \text{range} \\ \text{temperature} \end{bmatrix}$.

Answer

- $R_{11} = \sigma^2_{\text{range}}$: The variance of the radar's measurement noise, with units of **meters squared (m^2)**. It quantifies the uncertainty of the range measurement.

- $R_{22} = \sigma^2_{\text{temp}}$: The variance of the camera's measurement noise, with units of **Celsius squared (C^2)**. It quantifies the uncertainty of the temperature reading.

Pop Quiz 2: The Meaning of Off-Diagonals

The Standard Assumption

In the standard model, we define $\mathbf{R}$ as a diagonal matrix:

$$\mathbf{R} = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$

Question

Why are the off-diagonal elements set to zero? What would a non-zero value for R_{12} physically imply about the noise in your sensors?

Pop Quiz 2: The Meaning of Off-Diagonals

The Standard Assumption

In the standard model, we define $\mathbf{R}$ as a diagonal matrix:

$$\mathbf{R} = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$

Answer

The off-diagonal elements are zero because we assume the **measurement noises are statistically independent**.

- $R_{ij} = E[v_i v_j] = 0$ for $i \neq j$ is the mathematical statement for uncorrelated noise.
- A non-zero R_{12} would imply the noise sources are **correlated**. For example, a single faulty power supply could be causing simultaneous, related errors in both the radar and the camera, making their noise dependent on each other.

Pop Quiz 3: R's Influence on the Solution

The WLS Solution

$$\hat{\mathbf{x}} = (\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{R}^{-1}\mathbf{y}$$

Question

Imagine you replace your high-quality radar with a very cheap, noisy one. You are now much less confident in your range measurement compared to your temperature measurement.

How should you adjust the value of R_{11} (the range variance), and why does the $\mathbf{R}^{-1}$ term in the WLS equation make intuitive sense for this change?

Pop Quiz 3: R's Influence on the Solution

The WLS Solution

$$\hat{\mathbf{x}} = (\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{R}^{-1}\mathbf{y}$$

Answer

You should significantly **increase** the value of R_{11}.

- A large R_{11} signifies high variance, which means high uncertainty (low confidence) in the measurement.
- The $\mathbf{R}^{-1}$ term acts as a **weighting matrix**. When R_{11} is large, its corresponding diagonal term in $\mathbf{R}^{-1}$ (i.e., $1/R_{11}$) becomes very small.
- This small weight ensures the WLS solution "trusts" the noisy radar measurement less when calculating the final state estimate $\hat{\mathbf{x}}$.

The Problem with Batch Estimation

A major drawback for real-time systems:

The Inefficiency of the Batch Method

With every new measurement that arrives:

- The data matrices ($\mathbf{H}$ and $\mathbf{y}$) must grow in size.
- The entire solution must be re-computed from scratch using all past data.

Example: Imagine a satellite sending data once per second. After an hour (3600 measurements), you would need to re-process all 3600 points just to incorporate the next one. NOT SCALABLE.

How can we intelligently update our old estimate using only the new measurement, without having to re-process the entire data history every time?

Section 1.3: The Linear Recursive Estimator

Key Idea

Our estimate will not change significantly when a new measurement arrives. So, we can represent new estimate as a combination of old estimate and new measurement.

We can define our estimator in a **recursive** form:

The Recursive Estimator Form

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k-1})$$

- $\hat{\mathbf{x}}_{k-1}$ is our **previous estimate**.
- $\hat{\mathbf{x}}_k$ is our **new (updated) estimate**.
- The term $(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k-1})$ is the **correction term** or *innovation*. It's the difference between the actual measurement and our predicted measurement.
- $\mathbf{K}_k$ is the **gain matrix**, which determines how much we trust the correction term.

The Linear Recursive Estimator

The Recursive Estimator Form

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k-1})$$

- $\hat{\mathbf{x}}_{k-1}$ is our **previous estimate**.
- $\hat{\mathbf{x}}_k$ is our **new (updated) estimate**.
- The term $(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k-1})$ is the **correction term** or *innovation*. It's the difference between the actual measurement and our predicted measurement.
- $\mathbf{K}_k$ is the **gain matrix**, which determines how much we trust the correction term.

Our new goal is to find the optimal gain, $\mathbf{K}_k$, that minimizes our estimation error.

Next Few Steps

- Unbiased estimator recall.
- Showing why this form is powerful.
- How do we track uncertainty in the estimate ?
- How do we find K_k?

Quick Statistics Recall: What is an Unbiased Estimator?

In statistics, an **estimator** is a rule or formula used to guess an unknown **parameter** of a population using sample data. For example, we use the sample mean ($\bar{x}$) to estimate the true population mean (μ).

An estimator is called **unbiased** if its expected value (its average over many, many samples) is equal to the true parameter it's trying to estimate.

The Condition for Unbiasedness

An estimator $\hat{\theta}$ for a parameter θ is unbiased if:

$$E[\hat{\theta}] = \theta \quad \Longrightarrow \quad E[\theta - \hat{\theta}] = 0$$

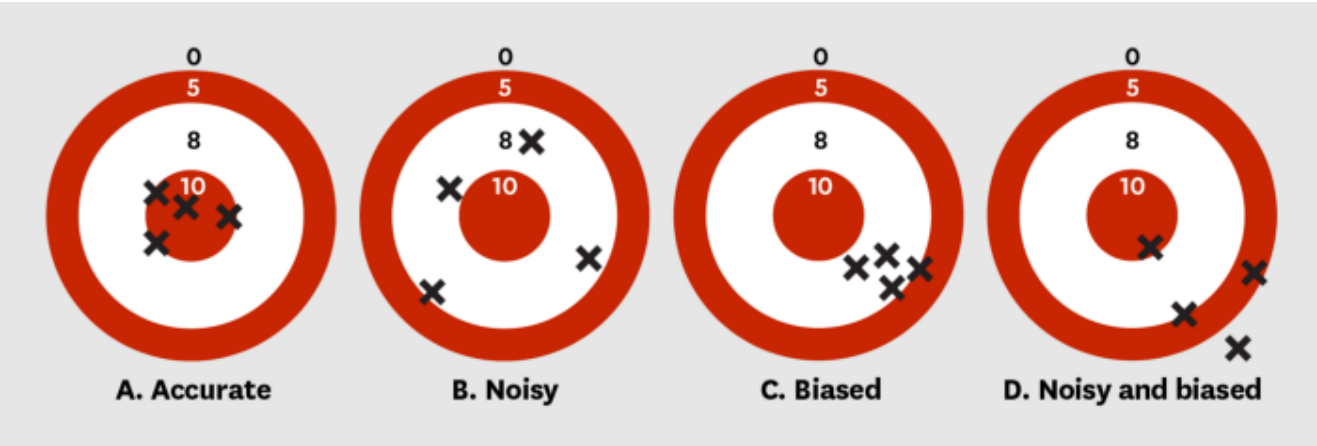
In simple terms, an unbiased estimator has no systematic error. It doesn't consistently overestimate or underestimate the true value.

Quick Statistics Recall: What is an Unbiased Estimator?

Analogy: The Archer

Imagine an archer shooting at a target. The bullseye is the true parameter.

- An **unbiased** archer's shots may spread around the bullseye, but their average position is the exact center.
- A **biased** archer is also trying to hit the bullseye, but their shots are systematically off; their average position might be high and to the right.



The Wisdom of the Crowd: An Unbiased Estimator

Social Experiment: Guess how many in the jar ?

The Wisdom of the Crowd: An Unbiased Estimator

- **Scenario:** A jar contains a true number of items, θ. A group of n people provides individual guesses, $g_1, g_2, \ldots, g_n$.
- **The Crowd's Estimate ($\hat{\theta}$):** We use the average of all guesses: $\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} g_i$
- **The Unbiased Assumption:** We assume individual guesses have random errors but are not systematically too high or too low: $E[g_i] = \theta$
- **The Result:** The expected value of the crowd's estimate is also the true value, making it an **unbiased estimator**. The individual errors cancel out.

$$E[\hat{\theta}] = E\left[\frac{1}{n}\sum g_i\right] = \frac{1}{n}\sum E[g_i] = \frac{1}{n}(n\theta) = \theta$$

Even if individual guesses are poor, their average converges to the truth.

Is The Crowd Always Unbiased?

The math works beautifully if individual guesses are truly unbiased. Random errors cancel out, and the average converges to the truth.

Do you think this is always the case ?

Why Recursive Form is Powerful? It is Unbiased!

Before $\mathbf{K}_k$, let's analyze the mean of the estimation error, $E[\mathbf{x} - \hat{\mathbf{x}}_k]$, to see if our recursive estimator is biased.

Derivation of the Error Mean

$$\begin{aligned} E[\mathbf{x} - \hat{\mathbf{x}}_k] &= E[\mathbf{x} - \{\hat{\mathbf{x}}_{k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k-1})\}] \\ &= E[\mathbf{x} - \hat{\mathbf{x}}_{k-1} - \mathbf{K}_k(\mathbf{H}_k\mathbf{x} + \mathbf{v}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k-1})] \\ &= E[(\mathbf{x} - \hat{\mathbf{x}}_{k-1}) - \mathbf{K}_k\mathbf{H}_k(\mathbf{x} - \hat{\mathbf{x}}_{k-1}) - \mathbf{K}_k\mathbf{v}_k] \\ &= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)E[\mathbf{x} - \hat{\mathbf{x}}_{k-1}] - \mathbf{K}_k E[\mathbf{v}_k] \end{aligned}$$

What Makes The Estimator Unbiased ?

If the measurement noise is zero-mean ($E[\mathbf{v}_k] = 0$) and the previous estimate was unbiased ($E[\mathbf{x} - \hat{\mathbf{x}}_{k-1}] = 0$), then the new error mean is also zero. This is a desirable property because it means that, on average, **our estimate will be correct**. Crucially, this holds true **regardless of our choice for the gain matrix $\mathbf{K}_k$**.

Conditions for an Unbiased Recursive Estimator

For our recursive estimator to be unbiased (meaning $E[\hat{\mathbf{x}}_k] = E[\mathbf{x}]$), two conditions must be met:

- **The Measurement Noise Must Be Unbiased**

  $$E[\mathbf{v}_k] = 0$$

  This is a fair assumption, and if a known sensor bias exists, we can subtract it from the measurements to achieve this.

- **The Initial Estimate Must Be Unbiased** Our previous estimate is unbiased ($E[\mathbf{x} - \hat{\mathbf{x}}_{k-1}] = 0$). Recursively roll this down to the initial case and we need:

  $$E[\mathbf{x} - \hat{\mathbf{x}}_0] = 0$$

  This means our initial guess must not have a systematic error. It does *not* mean our first guess $\hat{\mathbf{x}}_0$ has to be accurate or close to the true value, only that it's centered correctly on average.

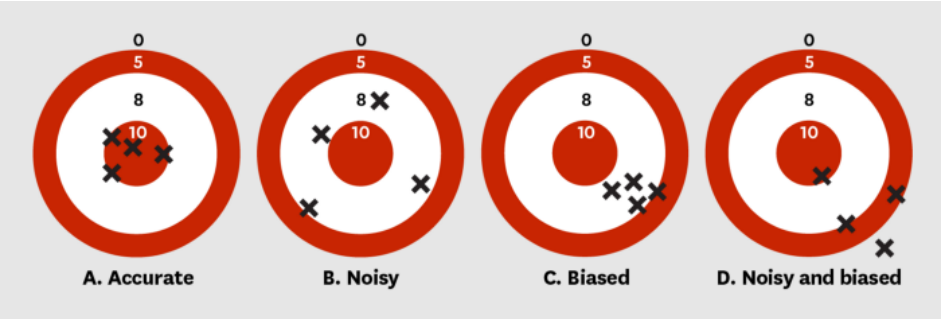
What if the initial guess is wrong?

We model its uncertainty. Wait for it!

Finding the Optimal Gain K_k

So, which gain is the best?

The optimal estimator is not just correct on average (unbiased), but is also the most **precise**.



This means it has the smallest possible error variance.

The New Optimality Criterion

We must find the gain $\mathbf{K}_k$ that minimizes the sum of the variances of the estimation errors.

$$J_k = E[(x_1 - \hat{x}_1)^2] + \cdots + E[(x_n - \hat{x}_n)^2]$$

Our goal is to choose $\mathbf{K}_k$ to make this total variance J_k as small as possible.

The Cost Function Optimization for K

Derivation of the Cost Function

$$\begin{aligned} J_k &= E[e_{x_1,k}^2 + \cdots + e_{x_n,k}^2] \\ &= E[\mathbf{e}_{x,k}^T \mathbf{e}_{x,k}] \\ &= E[\mathrm{Tr}(\mathbf{e}_{x,k}\mathbf{e}_{x,k}^T)] \quad \text{(Using the property } \mathbf{v}^T\mathbf{v} = \mathrm{Tr}(\mathbf{v}\mathbf{v}^T)) \\ &= \mathrm{Tr}(E[\mathbf{e}_{x,k}\mathbf{e}_{x,k}^T]) \end{aligned}$$

This leads us to a critical definition:

The Estimation-Error Covariance Matrix ($\mathbf{P}_k$)

$$\mathbf{P}_k = E[\mathbf{e}_{x,k}\mathbf{e}_{x,k}^T]$$

The diagonal elements of $\mathbf{P}_k$ are the variances of our estimation error.

Our goal is now simply to find the gain $\mathbf{K}_k$ that minimizes $\mathrm{Tr}(\mathbf{P}_k)$.

Deriving the Error Covariance Recursion

Goal: find a recursive formula for $\mathbf{P}_k = E[\mathbf{e}_{x,k}\mathbf{e}_{x,k}^T]$, and minimize.

Remember the recursive formula for the error:

$E[\mathbf{x} - \hat{\mathbf{x}}_k] = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)E[\mathbf{x} - \hat{\mathbf{x}}_{k-1}] - \mathbf{K}_k E[\mathbf{v}_k]$.

Expanding the Covariance Equation (Complicated)

$$\mathbf{P}_k = E\left[\left((\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{e}_{k-1} - \mathbf{K}_k\mathbf{v}_k\right)(\dots)^T\right]$$

$$\begin{aligned}
=&(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\underbrace{E[\mathbf{e}_{k-1}\mathbf{e}_{k-1}^T]}_{\mathbf{P}_{k-1}}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^T \\
&- (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)E[\mathbf{e}_{k-1}\mathbf{v}_k^T]\mathbf{K}_k^T \\
&- \mathbf{K}_k E[\mathbf{v}_k\mathbf{e}_{k-1}^T](\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^T \\
&+ \mathbf{K}_k \underbrace{E[\mathbf{v}_k\mathbf{v}_k^T]}_{\mathbf{R}_k}\mathbf{K}_k^T
\end{aligned}$$

Simplifying the Covariance Recursion

To simplify we assume that the estimation error at time $k-1$ ($\mathbf{e}_{k-1}$) is independent of the measurement noise at time k ($\mathbf{v}_k$). Therefore, their correlation is zero.

$$E[\mathbf{v}_k \mathbf{e}_{k-1}^T] = E[\mathbf{v}_k]E[\mathbf{e}_{k-1}^T] = \mathbf{0}$$

The two middle terms of our expanded equation disappear:

The Recursive Error Covariance Formula

This leaves us with the final recursive update for the error covariance:

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

This Equation Makes Sense

The new uncertainty ($\mathbf{P}_k$) increases if the previous uncertainty ($\mathbf{P}_{k-1}$) was high, or if the current measurement noise ($\mathbf{R}_k$) is high.

What P_k Really Tells Us

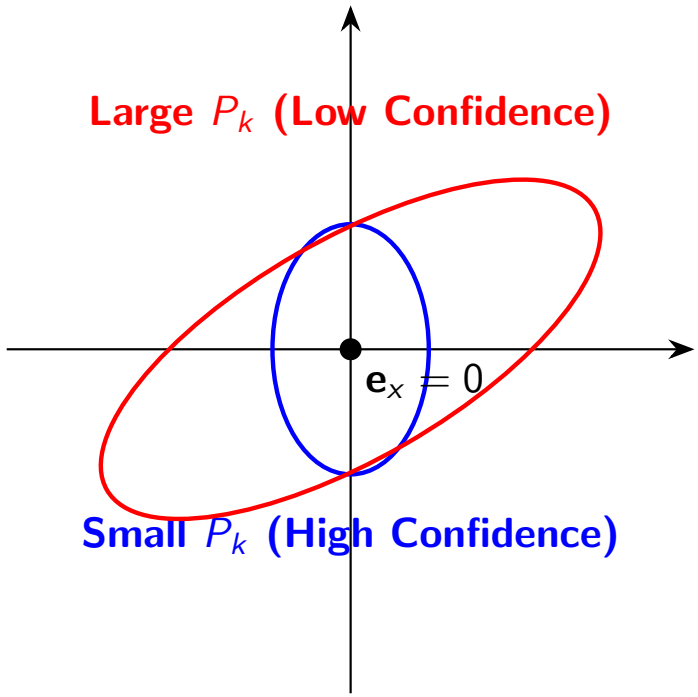
The matrix $\mathbf{P}_k = E[\mathbf{e}_{x,k}\mathbf{e}_{x,k}^T]$ is not just a mathematical term to be minimized. It is the filter's own measure of its **uncertainty** about the state estimate, $\hat{\mathbf{x}}_k$.

The Role of P_k

- **Diagonal elements ($P_k(i,i)$):** These are the variances (σ_i^2) of the error for each state. A smaller value means the filter is more confident about that specific state variable.
- **Off-diagonal elements ($P_k(i,j)$):** These are the covariances, indicating how the errors in different states are correlated.

Minimizing $\text{Tr}(\mathbf{P}_k)$ means minimizing the sum of these variances, which is equivalent to "shrinking" our overall uncertainty.

The Ellipse of Uncertainty



The Big Picture

Think of $\mathbf{P}_k$ as defining a "confidence ellipse" around our state estimate. A small, tight ellipse means the filter is very sure of its answer. A large, stretched-out ellipse means it's highly uncertain. The Kalman filter's magic lies in using this ellipse to intelligently incorporate new measurements.

Solving for the Optimal Gain K_k

GOAL: Find the gain $\mathbf{K}_k$ that minimizes the error variance, $J_k = \text{Tr}(\mathbf{P}_k)$.

Taking the Derivative of the Cost Function (Believe me)

Using the recursive covariance equation and rules of matrix calculus, the derivative is:

$$\frac{\partial J_k}{\partial \mathbf{K}_k} = 2(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k-1}(-\mathbf{H}_k^T) + 2\mathbf{K}_k\mathbf{R}_k = 0$$

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

Solving for the Optimal Gain K_k

GOAL: Find the gain $\mathbf{K}_k$ that minimizes the error variance, $J_k = \text{Tr}(\mathbf{P}_k)$.

Taking the Derivative of the Cost Function (Believe me)

Using the recursive covariance equation and rules of matrix calculus, the derivative is:

$$\frac{\partial J_k}{\partial \mathbf{K}_k} = 2(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k-1}(-\mathbf{H}_k^T) + 2\mathbf{K}_k\mathbf{R}_k = 0$$

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

Now, we have everything we need:

- Linear Recursive Estimator: $\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k-1})$
- Estimation Error Covariance (Joseph Form):
  $\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k-1}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T$
- Optimal Gain: $\mathbf{K}_k = \mathbf{P}_{k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k-1}\mathbf{H}_k^T + \mathbf{R}_k)^{-1}$

The Recursive Least Squares (RLS) Algorithm

1. System Model

- $\mathbf{y}_k = \mathbf{H}_k\mathbf{x} + \mathbf{v}_k$
- State $\mathbf{x}$ is constant.
- Zero-mean and uncorrelated noise: $E[\mathbf{v}_k] = 0$ and $E[\mathbf{v}_k\mathbf{v}_j^T] = \mathbf{R}_k\delta_{k-j}$.

The Recursive Least Squares (RLS) Algorithm

1. System Model

- $\mathbf{y}_k = \mathbf{H}_k\mathbf{x} + \mathbf{v}_k$
- State $\mathbf{x}$ is constant.
- Zero-mean and uncorrelated noise: $E[\mathbf{v}_k] = 0$ and $E[\mathbf{v}_k\mathbf{v}_j^T] = \mathbf{R}_k\delta_{k-j}$.

2. Initialization (at k=0)

- Initial state estimate: $\hat{\mathbf{x}}_0 = E[\mathbf{x}]$
- Initial error covariance: $\mathbf{P}_0 = E[(\mathbf{x} - \hat{\mathbf{x}}_0)(\mathbf{x} - \hat{\mathbf{x}}_0)^T]$

The Recursive Least Squares (RLS) Algorithm

1. System Model

- $\mathbf{y}_k = \mathbf{H}_k\mathbf{x} + \mathbf{v}_k$
- State $\mathbf{x}$ is constant.
- Zero-mean and uncorrelated noise: $E[\mathbf{v}_k] = 0$ and $E[\mathbf{v}_k\mathbf{v}_j^T] = \mathbf{R}_k\delta_{k-j}$.

2. Initialization (at k=0)

- Initial state estimate: $\hat{\mathbf{x}}_0 = E[\mathbf{x}]$
- Initial error covariance: $\mathbf{P}_0 = E[(\mathbf{x} - \hat{\mathbf{x}}_0)(\mathbf{x} - \hat{\mathbf{x}}_0)^T]$

3. The Recursive Loop (for k = 1, 2, ...)

1. **Compute the Gain:** $\mathbf{K}_k = \mathbf{P}_{k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k-1}\mathbf{H}_k^T + \mathbf{R}_k)^{-1}$
2. **Update the State Estimate:** $\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k-1})$
3. **Update the Error Covariance:** $\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k-1}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T$

Recursive Least Squares (RLS): Key Takeaways

What It Is...

An **online algorithm** that updates its estimate every time a new measurement.

It performs the same job as "batch" least squares, but it does so sequentially, one measurement at a time.

Why It's Useful...

- **It's Efficient:** It avoids re-processing the entire history.
- **It Needs Minimal Memory:** It doesn't need to store all past data points, only the last state estimate and its uncertainty.
- **It's Ideal for Real-Time Systems:** It's perfectly suited for applications with streaming data, such as navigation, system identification, and adaptive filtering.

Congratulations!

You have now derived the complete set of equations for the **Recursive Least Squares (RLS) algorithm**.

What's Covered

- The System Model and Initialization Steps
- The Optimal Gain Equation ($\mathbf{K}_k$)
- The State Update Equation ($\hat{\mathbf{x}}_k$)
- The Error Covariance Update Equation ($\mathbf{P}_k$)

This is the half of it!

You are half way there for Kalman Filter!

Least Squares: The Foundation of Modern AI

The idea of minimizing squared error is the fundamental building block for machine learning.

- **Curve Fitting:** Finds the "best-fit" line by minimizing the squared distance (error) to the data points.
- **Machine Learning:** Generalizes this. "Training" a model means finding parameters that minimize a **loss function**, which is just a measure of total error.
- **Deep Learning:** Takes this to the extreme. (Going **Deep**!) Training a neural network is a massive optimization problem tweaking millions of parameters to make the loss as small as possible.

The Core Principle is Always the Same

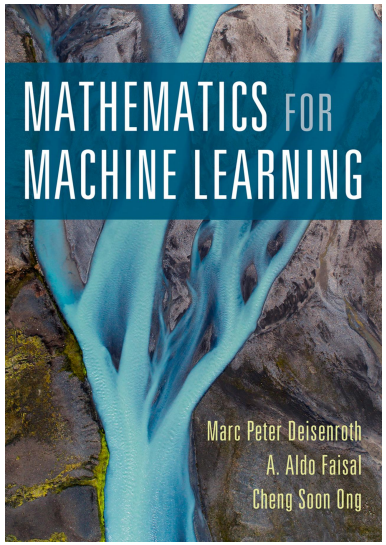
Find the parameters that make the error as small as possible.
Understanding least squares is understanding the heart of how AI models learn.

For Further Reading

If you want to build a deep understanding of the mathematical principles behind least squares and machine learning, this book is an outstanding resource.

Recommended Textbook

Deisenroth, Marc Peter, A. Aldo Faisal, and Cheng Soon Ong. **Mathematics for Machine Learning**. Cambridge University Press, 2020.



Why This Book?

Best of all, a **free online version** is provided by the authors at: **mml-book.com**

Code Review 1

Batch Least Square Estimator v. Recursive Least Square Estimator

Step 1: Initialization

- **Initial Guess ($\hat{\mathbf{x}}_0$):** Our first estimate: $\hat{\mathbf{x}}_0 = [0.0]$ However, the true value is 4.2.
- **Initial Covariance ($\mathbf{P}_0$):** Our uncertainty: $\mathbf{P}_0 = [100.0]$

Step 2: The Recursive Loop

- **Measurement Model:**

$$y_k = H_k \mathbf{x} + v_k$$

- **Example Setup:** Measure the state directly ($H_k = 1.0$) with significant noise (variance $R = 100.0$).

The core idea: Start with high uncertainty (P_0 is large) and let each new measurement y_k reduce that uncertainty over time.

Chapter 2: Dynamic Transition of Variables

From Static to Dynamic: The Next Frontier

The Recursive Least Squares (RLS) algorithm is a powerful tool for estimating an unknown **constant** that restricts its use in many real-world problems:

The "Constant State" Assumption

RLS assumes the true value we are estimating, $\mathbf{x}$, never changes.

But the Real World is Dynamic...

Most systems we truly care about are constantly changing:

- The position and velocity of a moving car
- The orientation of a flying drone
- The temperature and pressure in a weather system

Dynamical Systems

We now consider systems where the state $\mathbf{x}$ is allowed to change over time. A linear discrete-time system can be modeled as follows:

The State Equation

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}$$

- $\mathbf{F}_{k-1}$ is the **state transition matrix** (describes how the system evolves on its own). Might be derived from equations of motion.
- $\mathbf{G}_{k-1}\mathbf{u}_{k-1}$ is the known **control input**.
- $\mathbf{w}_{k-1}$ is the **process noise** (unpredictable disturbances).

Toy Examples of Linear System Modeling

State Transition Matrix (F)

Cart with constant velocity, no external forces.

State Vector: $\mathbf{x} = \begin{bmatrix} p \\ v \end{bmatrix}$

Physics of Motion: Time step Δt:

$$p_k = p_{k-1} + v_{k-1} \cdot \Delta t$$
$$v_k = v_{k-1}$$

Matrix Form:

$$\underbrace{\begin{bmatrix} p_k \\ v_k \end{bmatrix}}_{\mathbf{x}_k} = \underbrace{\begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}}_{\mathbf{F}} \underbrace{\begin{bmatrix} p_{k-1} \\ v_{k-1} \end{bmatrix}}_{\mathbf{x}_{k-1}}$$

Control Matrix (G)

A thruster with a known acceleration, a.

Control Input: $\mathbf{u} = [a]$

Physics with Control:

$$p_k = p_{k-1} + v_{k-1}\Delta t + \frac{1}{2} a_{k-1} \Delta t^2$$
$$v_k = v_{k-1} + a_{k-1}\Delta t$$

Matrix Form:

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \underbrace{\begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix}}_{\mathbf{G}} \underbrace{[a_{k-1}]}_{\mathbf{u}_{k-1}}$$

The Things We Can't Model

What is Process Noise ($\mathbf{w}$)?

It represents all the small, unpredictable forces that affect the state.

- For a drone : Unexpected wind gusts or air pockets.
- For a rover : Unpredictable road bumps or tire slippage.
- For a robot arm : Tiny vibrations in the motors.

Properties of Process Noise (Are these realistic ?)

We model $\mathbf{w}_k$ as zero-mean, **white** Gaussian noise:

- $E[\mathbf{w}_k] = 0$ (The noise doesn't systematically push in one direction).
- $E[\mathbf{w}_k \mathbf{w}_j^T] = \mathbf{Q}_k \delta_{kj}$ (The noise is uncorrelated in time; $\mathbf{Q}_k$ is its covariance).

The **Process Noise Covariance** $\mathbf{Q}_k$ represents the amount of uncertainty added to the system at each time step.

Process Noise Assumptions: Feasibility

The White Noise Assumptions

- $E[\mathbf{w}_k] = \mathbf{0}$
  - The implicit assumption $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$ is often a good approximation (via Central Limit Theorem).
- $E[\mathbf{w}_k \mathbf{w}_j^T] = \mathbf{Q}_k \delta_{kj}$, δ_{kj} is 1 if $k = j$ otherwise 0.
  - A crucial simplification. It makes the state a Markov process, which is essential for the standard filter's derivation. $\mathbf{Q}_k$ becomes a tuning parameter. Most physical noise is "colored" (temporally correlated).
  - **Wheel Slip & Scrub:** Uneven terrain, sticky spots on the floor, or tire deflation cause effects that persist over several timesteps. A slip doesn't just happen instantaneously; it affects motion for a short duration.

Process Noise Assumptions: Feasibility

The White Noise Assumptions

- $E[\mathbf{w}_k] = \mathbf{0}$
  - $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$ (Central Limit Theorem)
- $E[\mathbf{w}_k \mathbf{w}_j^T] = \mathbf{Q}_k \delta_{kj}$, δ_{kj} is 1 if $k = j$ otherwise 0.
  - Most physical noise is "colored" (temporally correlated). $\mathbf{Q}_k$ becomes a tuning parameter.

Cost of Colored Noise

- **Suboptimal Performance:** If the noise is strongly colored and this is ignored, filter may become overconfident.
- **The Role of Q:** In practice, roboticists *inflate* the process noise covariance matrix Q to compensate for these unmodeled, colored noise effects. This prevents filter overconfidence but is an ad-hoc solution that results in a less accurate (suboptimal) state estimate.

Statistics of Dynamical Systems

The State $\mathbf{x}$ is a Random Variable (Just like $\mathbf{w}$!)

Because of random process noise, we can never know the true state $\mathbf{x}_k$ perfectly. Instead, we treat it as a random variable and track its statistical properties.

Mean (Our Estimate)

- Our best estimate of the state.
- This is the value we store and use in practice.

$$\hat{\mathbf{x}}_k = E[\mathbf{x}_k]$$

Covariance (Our Uncertainty)

- Represents the uncertainty or "spread" of our estimate.
- This is why we must always track the error covariance matrix $\mathbf{P}_k$.

$$\mathbf{P}_k = E[(\mathbf{x}_k - \hat{\mathbf{x}}_k)(\mathbf{x}_k - \hat{\mathbf{x}}_k)^T]$$

Propagation of Random Variable: State Mean

By taking the expected value of both sides and leveraging the linearity of expectation:

$$\begin{aligned}\bar{\mathbf{x}}_k &= E[\mathbf{x}_k] \\ &= E[\mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}] \\ &= \mathbf{F}_{k-1}E[\mathbf{x}_{k-1}] + \mathbf{G}_{k-1}E[\mathbf{u}_{k-1}] + E[\mathbf{w}_{k-1}]\end{aligned}$$

Assuming:

- The control input $\mathbf{u}_{k-1}$ is deterministic ($E[\mathbf{u}_{k-1}] = \mathbf{u}_{k-1}$)
- Process noise is zero-mean ($E[\mathbf{w}_{k-1}] = \mathbf{0}$)

, we get the state prediction:

$$\bar{\mathbf{x}}_k = \mathbf{F}_{k-1}\bar{\mathbf{x}}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1}$$

Propagation of Random Variable: State Covariance

Part 1: Expansion

Next, how does the uncertainty of our state, $\mathbf{P}_k$, change with time?
We start with the definition of the state covariance:
$\mathbf{P}_k = E[(\mathbf{x}_k - \bar{\mathbf{x}}_k)(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T]$.

First, let's find the expression for the error term, $(\mathbf{x}_k - \bar{\mathbf{x}}_k)$:

$$(\mathbf{x}_k - \bar{\mathbf{x}}_k) = (\mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}) - (\mathbf{F}_{k-1}\bar{\mathbf{x}}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1})$$

$$= \mathbf{F}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1}) + \mathbf{w}_{k-1}$$

Expanding $(\mathbf{x}_k - \bar{\mathbf{x}}_k)(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T$

$$\begin{aligned}
&= (\mathbf{F}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1}) + \mathbf{w}_{k-1})\,(\mathbf{F}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1}) + \mathbf{w}_{k-1})^T \\
&= (\mathbf{F}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1}) + \mathbf{w}_{k-1})\,((\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})^T\mathbf{F}_{k-1}^T + \mathbf{w}_{k-1}^T) \\
&= \mathbf{F}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})^T\mathbf{F}_{k-1}^T + \mathbf{F}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})\mathbf{w}_{k-1}^T \\
&\quad + \mathbf{w}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})^T\mathbf{F}_{k-1}^T + \mathbf{w}_{k-1}\mathbf{w}_{k-1}^T
\end{aligned}$$

Propagation of Random Variable: State Covariance

Part 2: The Simplification

Expanding $(\mathbf{x}_k - \bar{\mathbf{x}}_k)(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T$ cont'd

$$
\begin{aligned}
=&\mathbf{F}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})^T\mathbf{F}_{k-1}^T + \mathbf{F}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})\mathbf{w}_{k-1}^T \\
&+ \mathbf{w}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})^T\mathbf{F}_{k-1}^T + \mathbf{w}_{k-1}\mathbf{w}_{k-1}^T
\end{aligned}
$$

The state error at time $k-1$, $(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})$, is independent of the process noise at time $k-1$, $\mathbf{w}_{k-1}$. Because it affects next state $\mathbf{x}_k$
Therefore, the expected value of their cross-products is zero:

$$E[\mathbf{F}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})\mathbf{w}_{k-1}^T] = 0$$

This makes the two middle "cross terms" in our expansion disappear.

Propagation of Random Variable: State Covariance

Part 3: The Result

The Covariance Propagation Equation

This leaves us with the final, clean equation for how uncertainty evolves:

$$\mathbf{P}_k = E[\mathbf{F}_{k-1}(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})^T \mathbf{F}_{k-1}^T + \mathbf{w}_{k-1}\mathbf{w}_{k-1}^T]$$

$$\mathbf{P}_k = \mathbf{F}_{k-1} \underbrace{E[(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1})(\dots)^T]}_{\mathbf{P}_{k-1}} \mathbf{F}_{k-1}^T + \underbrace{E[\mathbf{w}_{k-1}\mathbf{w}_{k-1}^T]}_{\mathbf{Q}_{k-1}}$$

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

Intuition

The new uncertainty is the old uncertainty transformed by the system dynamics, plus some new uncertainty from the process noise.

Putting All Dynamic Propagation Together

1. State Mean Propagation

$$\bar{\mathbf{x}}_k = \mathbf{F}_{k-1}\bar{\mathbf{x}}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1}$$

2. State Covariance Propagation

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

Other Forms of Process Noise Handling

Some resources represent process $\mathbf{w}$ noise multiplied by some other matrix $\mathbf{L}$:

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{G}\mathbf{u}_{k-1} + \mathbf{L}\tilde{\mathbf{w}}_{k-1}$$

This is equivalent to a standard model with a new noise term $\mathbf{w}_k = \mathbf{L}\tilde{\mathbf{w}}_{k-1}$ and a transformed covariance $\mathbf{Q}$ and it can be easily converted to our form:

$$\mathbf{Q} = E[(\mathbf{L}\tilde{\mathbf{w}})(\mathbf{L}\tilde{\mathbf{w}})^T] = \mathbf{L}\tilde{\mathbf{Q}}\mathbf{L}^T$$

Example: A Predator-Prey Ecosystem Model

We model the population of a predator, $x(1)$, and its prey, $x(2)$.

- The **predator** population decreases due to overcrowding but increases with the availability of prey.
- The **prey** population decreases due to predators but increases due to an external food supply.

Linear System of Equations

We model the population of a predator, $x_k(1)$, and its prey, $x_k(2)$, at time step k. The population changes are described by:

$$x_{k+1}(1) = x_k(1) - 0.8x_k(1) + 0.4x_k(2) + w_k(1)$$
$$x_{k+1}(2) = x_k(2) - 0.4x_k(1) + u_k + w_k(2)$$

The populations are also subject to random disturbances $\mathbf{w}_k$ due to environmental factors.

Example: A Predator-Prey Ecosystem Model

Linear System of Equations

$$x_{k+1}(1) = x_k(1) - 0.8x_k(1) + 0.4x_k(2) + w_k(1)$$
$$x_{k+1}(2) = x_k(2) - 0.4x_k(1) + u_k + w_k(2)$$

The State-Space Model: $\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}u_k + \mathbf{w}_k$

The individual dynamics can be written in matrix form:

$$\underbrace{\begin{bmatrix} x_{k+1}(1) \\ x_{k+1}(2) \end{bmatrix}}_{\mathbf{x}_{k+1}} = \underbrace{\begin{bmatrix} 0.2 & 0.4 \\ -0.4 & 1 \end{bmatrix}}_{\mathbf{F}} \underbrace{\begin{bmatrix} x_k(1) \\ x_k(2) \end{bmatrix}}_{\mathbf{x}_k} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{\mathbf{G}} u_k + \mathbf{w}_k$$

where the process noise $\mathbf{w}_k$ has a covariance matrix $\mathbf{Q} = \text{diag}(1, 2)$.

Predator-Prey: Simulating the Population Mean

Initial State ($\mathbf{x}_0$)

10 predators, 20 prey

$$\mathbf{x}_0 = \begin{bmatrix} 10 \\ 20 \end{bmatrix}$$

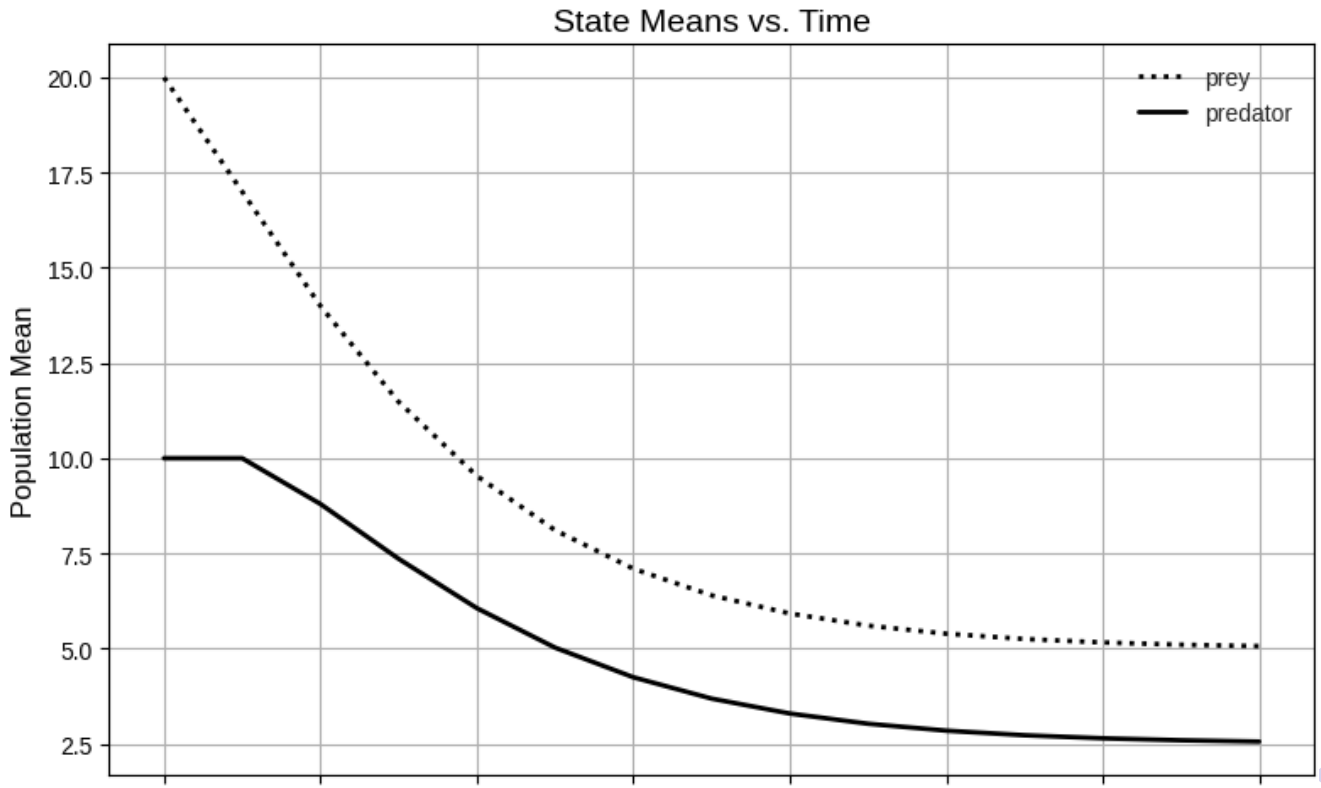
Initial Covariance ($\mathbf{P}_0$)

$$\mathbf{P}_0 = \begin{bmatrix} 10 & 0 \\ 0 & 40 \end{bmatrix}$$

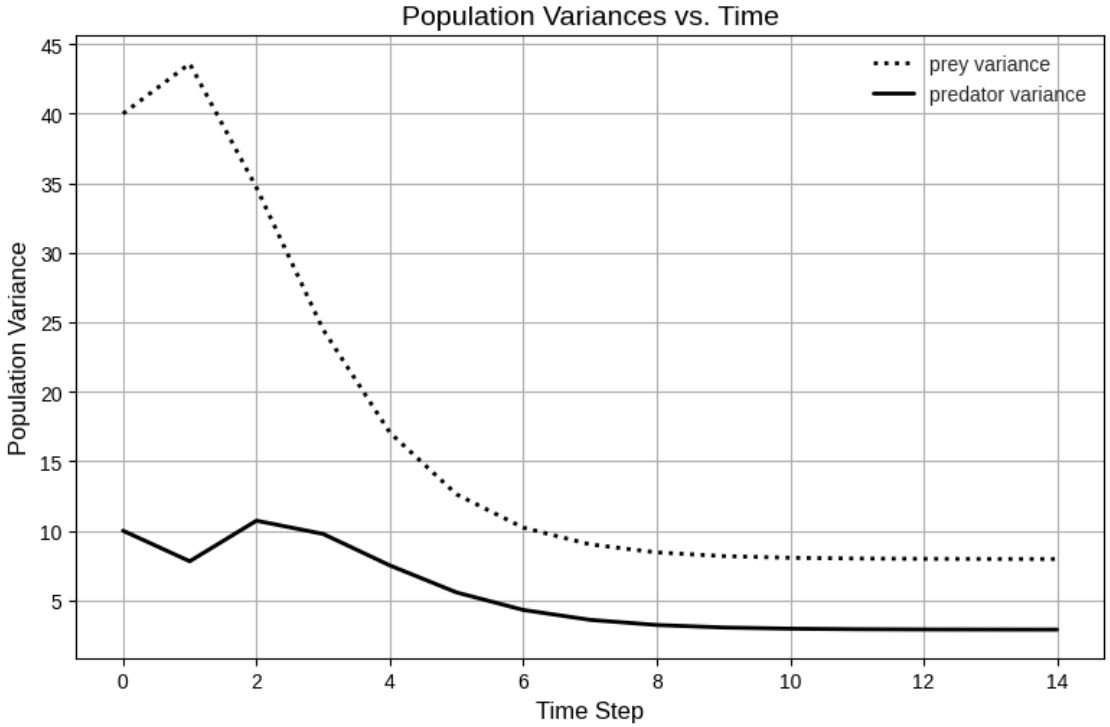
Control Input ($\mathbf{u}_k$)

Constant food for prey

$$u_k = 1$$



Predator-Prey: Uncertainty and Steady State



Steady-State Solution

Mean ($\bar{x}_{ss}$): $\bar{\mathbf{x}}_{ss} \approx \begin{bmatrix} 2.5, 5 \end{bmatrix}^T$

Covariance (P_{ss}):

$$\mathbf{P}_{ss} \approx \begin{bmatrix} 2.88 & 3.08 \\ 3.08 & 7.96 \end{bmatrix}$$

Congratulations!

You now understand how to predict the evolution of a linear system, accounting for its internal dynamics and the effects of random noise.

What's Covered

- Discrete-Time Linear System Dynamics
- State Transition ($\mathbf{F}$) Control ($\mathbf{G}$)
- Process Noise ($\mathbf{w}$, $\mathbf{Q}$)
- State as a Random Variable
- Propagation of Mean ($\hat{\mathbf{x}}$)
- Propagation of Covariance ($\mathbf{P}$)

This is the other half of the Kalman Filter!

We will put everything into together, now!

Chapter 3: The Kalman Filter

The Optimal Estimation Algorithm

The Core Idea: A Two-Step Cycle

Invented by Rudolf Kálmán in 1960, the filter optimally estimates the state of a dynamic system.
It is basically combination of two things you just learned:

- **Dynamic System Propagation** Forecast the next state using the system's dynamic model.
- **Recursive Least Square Optimization:** Correct the forecast using a new, noisy measurement.

History

Rumors say Thorvald Nicolai Thiele and Peter Swerling developed a similar algorithm earlier (1958). However, Kalman was inspired to derive the Kalman filter by applying state variables to the Wiener filtering problem, which is a RLS based filter to estimate state.

Mathematical Model: Discrete-Time Linear KF

The System Equations

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \quad \text{(Propagation Model)}$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \quad \text{(Measurement Model - RLS)}$$

Statistical Assumptions for Noise

- **Process Noise:** $\mathbf{w}_k \sim (\mathbf{0}, \mathbf{Q}_k)$ $E[\mathbf{w}_k\mathbf{w}_j^T] = \mathbf{Q}_k\delta_{kj}$
- **Measurement Noise:** $\mathbf{v}_k \sim (\mathbf{0}, \mathbf{R}_k)$ $E[\mathbf{v}_k\mathbf{v}_j^T] = \mathbf{R}_k\delta_{kj}$
- **Uncorrelated Noise Sources:** $E[\mathbf{w}_k\mathbf{v}_j^T] = \mathbf{0}$ for all k, j

δ_{kj} is 1 if $k = j$ otherwise 0. (Kronecker delta function)

Our Goal

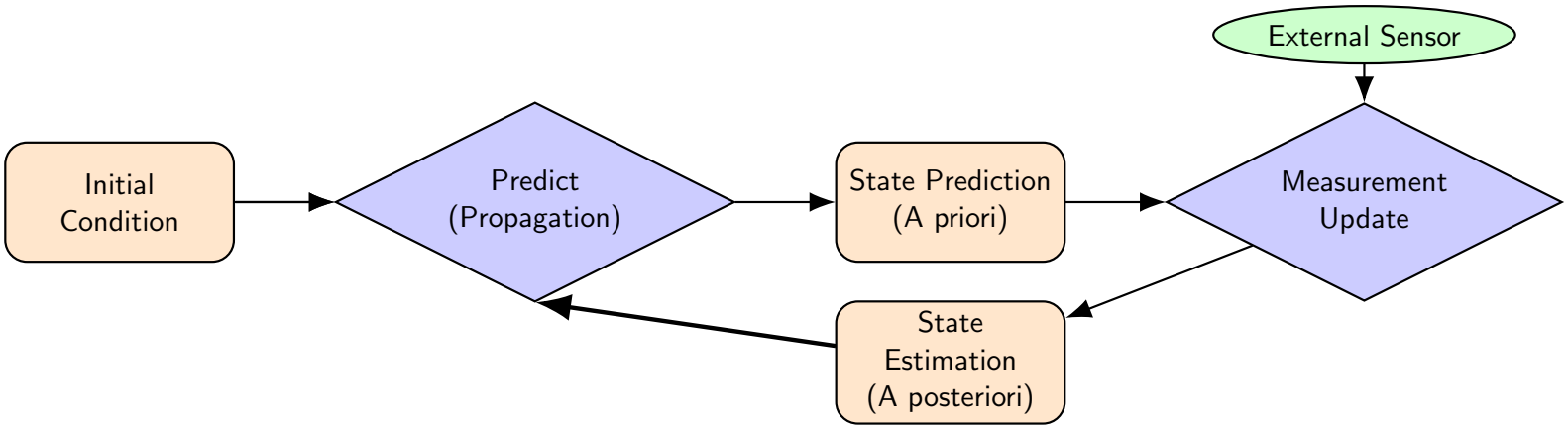
Find the best estimate for the state $\mathbf{x}_k$ given system dynamics $(\mathbf{F}, \mathbf{G}, \mathbf{H})$ and noisy measurements $\mathbf{y}_k$.

The Predict-Update Cycle

The Core Idea

Our state propagation model allows us to **predict** the future state. However, due to noise, this prediction will naturally drift from the true state over time.

To correct this drift, we use a new measurement, $\mathbf{y}_k$, to **update** our prediction. The final estimate is a blend of the prediction and the information from the new measurement.



A Priori vs. A Posteriori Estimates

Distinguishing Between Estimates

It's crucial to distinguish between the estimate *before* and *after* the measurement update. We use a superscript notation for this:

A Priori Prediction ($\hat{\mathbf{x}}_k^-$)

Prediction of the state at time k **before** the measurement at time k.

$$\hat{\mathbf{x}}_k^- = E[\mathbf{x}_k | \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{k-1}]$$

A Posteriori Estimate ($\hat{\mathbf{x}}_k^+$)

Updated estimate **after** incorporating the measurement $\mathbf{y}_k$.

$$\hat{\mathbf{x}}_k^+ = E[\mathbf{x}_k | \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k]$$

State Estimates on a Timeline

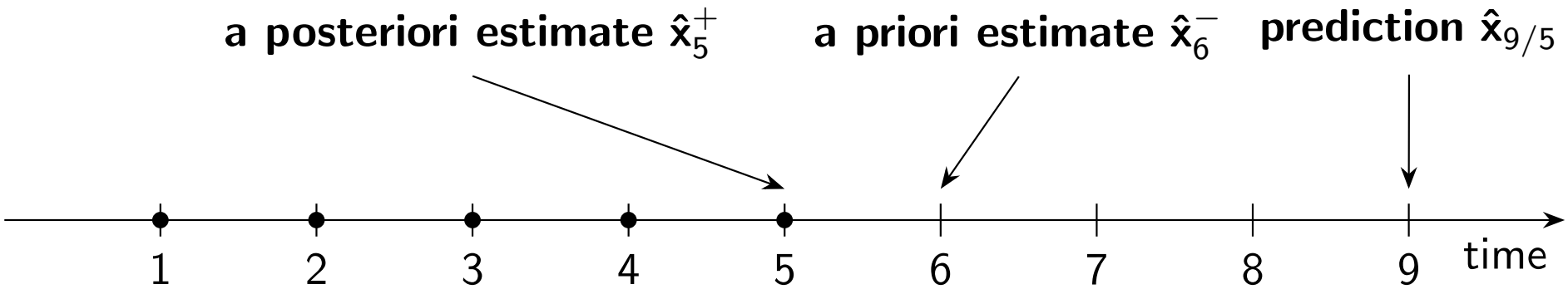


Figure: A timeline showing the relationship between different types of state estimates, assuming measurements are available up to and including time $k = 5$.

Initial Estimate and Error Covariance

The Starting Point: The Initial Estimate ($\hat{\mathbf{x}}_0^+$)

$\hat{\mathbf{x}}_0^+$ initial estimate of the state $\mathbf{x}_0$ *before* any measurements.
Since no measurement data is available, it is reasonable to form this estimate as the expected value of the initial state:

$$\hat{\mathbf{x}}_0^+ = E[\mathbf{x}_0]$$

The Error Covariance Matrix ($\mathbf{P}_k$)

- **A Priori Error Covariance ($\mathbf{P}_k^-$):** Based on our prediction, $\hat{\mathbf{x}}_k^-$.

$$\mathbf{P}_k^- = E[(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)^T]$$

- **A Posteriori Error Covariance ($\mathbf{P}_k^+$):** Based on estimate, $\hat{\mathbf{x}}_k^+$.

$$\mathbf{P}_k^+ = E[(\mathbf{x}_k - \hat{\mathbf{x}}_k^+)(\mathbf{x}_k - \hat{\mathbf{x}}_k^+)^T]$$

First Few Steps

Step 1: Initialization

Initialize

State Mean:

$$\hat{\mathbf{x}}_0^+ = E[\mathbf{x}_0]$$

State Covariance:

$$\mathbf{P}_0^+ = E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0^+)(\mathbf{x}_0 - \hat{\mathbf{x}}_0^+)^T]$$

- If we know the initial state perfectly: $\mathbf{P}_0^+ = \mathbf{0}$.
- If we have no idea about the initial state: $\mathbf{P}_0^+ = \infty\mathbf{I}$ (in practice, a very large number).

Step 2: Prediction (Propagation Equations)

Propagating from k=0 to the First A Priori Estimate at k=1

Predicting the State

The state propagation equation:

$$\hat{\mathbf{x}}_1^- = \mathbf{F}_0 \hat{\mathbf{x}}_0^+ + \mathbf{G}_0 \mathbf{u}_0$$

Predicting the Covariance

The covariance propagation equation:

$$\mathbf{P}_1^- = \mathbf{F}_0 \mathbf{P}_0^+ \mathbf{F}_0^T + \mathbf{Q}_0$$

Just Like We Did in Dynamic Propagation

We simply push our initial estimate and its uncertainty through the system dynamics to see where we expect to be at the next time step, *before* we get a new measurement.

General Time Prediction Equations

Prediction in General

From our updated estimate at $k-1$, we predict the state and covariance at time k:

State Prediction:

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

Covariance Prediction:

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

Similarity to RLS

Notation Relationships

Least Squares Estimation		Kalman Filtering
$\hat{\mathbf{x}}_{k-1}$ = estimate before $\mathbf{y}_k$	$\Longrightarrow$	$\hat{\mathbf{x}}_k^-$ = a priori estimate
$\mathbf{P}_{k-1}$ = covariance before $\mathbf{y}_k$	$\Longrightarrow$	$\mathbf{P}_k^-$ = a priori covariance
$\hat{\mathbf{x}}_k$ = estimate after $\mathbf{y}_k$	$\Longrightarrow$	$\hat{\mathbf{x}}_k^+$ = a posteriori estimate
$\mathbf{P}_k$ = covariance after $\mathbf{y}_k$	$\Longrightarrow$	$\mathbf{P}_k^+$ = a posteriori covariance

Step 3: Update (RLS Equations)

Estimate First A Posteriori Estimate at k=1

The Measurement Update Equations

1. **Compute the Kalman Gain (K_1):** Determines how much to trust the measurement.

$$\mathbf{K}_1 = \mathbf{P}_1^- \mathbf{H}_k^T (\mathbf{H}_1 \mathbf{P}_1^- \mathbf{H}_1^T + \mathbf{R}_1)^{-1}$$

2. **Update the State Estimate ($\hat{x}_1^+$):** Correct the prediction with the measurement residual.

$$\hat{\mathbf{x}}_1^+ = \hat{\mathbf{x}}_1^- + \mathbf{K}_1 (\mathbf{y}_1 - \mathbf{H}_1 \hat{\mathbf{x}}_1^-)$$

3. **Update the Covariance (P_1^+):** Reduce the uncertainty based on the measurement's information.

$$\mathbf{P}_1^+ = (\mathbf{I} - \mathbf{K}_1 \mathbf{H}_1) \mathbf{P}_1^- (\mathbf{I} - \mathbf{K}_1 \mathbf{H}_1)^T + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^T$$

General Time Estimation Equations

Estimate in General

The logic is identical to the Recursive Least Squares development. Recall the RLS update equations for a constant state:

$$\begin{aligned}
\mathbf{K}_k &= \mathbf{P}_{k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k-1}\mathbf{H}_k^T + \mathbf{R}_k)^{-1} \\
\hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_{k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k-1}) \\
\mathbf{P}_k &= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k-1}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T
\end{aligned}$$

where $\hat{\mathbf{x}}_{k-1}$ and $\mathbf{P}_{k-1}$ were our estimates *before* processing $\mathbf{y}_k$.

RLS Analogy

To get the Kalman Filter measurement update equations, we simply replace the RLS notation with our more precise *a priori* and *a posteriori* notation.

The Complete Discrete-Time Kalman Filter Algorithm

1. The System Model

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}, \quad \mathbf{w}_k \sim N(\mathbf{0}, \mathbf{Q}_k)$$
$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k, \quad \mathbf{v}_k \sim N(\mathbf{0}, \mathbf{R}_k)$$

2. Initialization (at k=0)

$$\hat{\mathbf{x}}_0^+ = E[\mathbf{x}_0]$$
$$\mathbf{P}_0^+ = E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0^+)(\mathbf{x}_0 - \hat{\mathbf{x}}_0^+)^T]$$

The Complete Discrete-Time Kalman Filter Algorithm

3. The Recursive Loop (for k = 1, 2, ...)

Predict

- *Project the state ahead:* $\hat{\mathbf{x}}_k^- = \mathbf{F}_{k-1}\hat{\mathbf{x}}_{k-1}^+ + \mathbf{G}_{k-1}\mathbf{u}_{k-1}$
- *Project the covariance ahead:* $\mathbf{P}_k^- = \mathbf{F}_{k-1}\mathbf{P}_{k-1}^+\mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1}$

Update

- *Compute Kalman Gain:* $\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^T + \mathbf{R}_k)^{-1}$
- *Update estimate with measurement:* $\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-)$
- *Update the covariance:* $\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_k^-(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T$

Alternative Formulations

There are many alternative equations to these. They are derived with different motivations like computational efficiency. However, the ones above are computationally robust.

Offline Computation

An Important Practical Advantage: Offline Computation

The covariance matrices $(\mathbf{P}_k^-, \mathbf{P}_k^+)$ and the Kalman gain $(\mathbf{K}_k)$ depend only on the system model $(\mathbf{F}, \mathbf{H}, \mathbf{Q}, \mathbf{R})$, **not on the measurements $\mathbf{y}_k$**.
This has two major benefits:

- **Real-Time Efficiency:** The gain $\mathbf{K}_k$ can be pre-computed offline and stored. During real-time operation, only the state estimate equations need to be run, drastically reducing the computational load. This is critical for embedded systems.
- **Offline Performance Analysis:** Since the error covariance $\mathbf{P}_k$ (which indicates filter accuracy) can be computed offline, the filter's performance can be evaluated and tuned before it is ever deployed on the real system.

Note: This is a unique advantage of the linear KF. For nonlinear filters, the gain generally depends on the measurements and must be computed in real-time.

Why is the Kalman Filter "Optimal"?

The Estimation Error

We define the error as:

$$\tilde{\mathbf{x}}_k = \mathbf{x}_k - \hat{\mathbf{x}}_k$$

The "Unbiased" Property

$$E[\hat{\mathbf{x}}_k] = E[\mathbf{x}_k] \quad \Longrightarrow \quad E[\tilde{\mathbf{x}}_k] = \mathbf{0}$$

The Kalman filter is designed to satisfy this condition.

The "Best" Property

The "best" estimator minimizes the error covariance:

$$\min E[\tilde{\mathbf{x}}_k^T \mathbf{S}_k \tilde{\mathbf{x}}_k]$$

where $\mathbf{S}_k$ is a positive definite weighting matrix. This is equivalent to minimizing our uncertainty.

Kalman Filter Optimality Conditions

The Best *Linear* Unbiased Estimator (BLUE)

If the following conditions hold:

- The system model is linear.
- Process noise ($\mathbf{w}_k$) and measurement noise ($\mathbf{v}_k$) are **zero-mean**, **white**, and **uncorrelated** with each other.

Conclusion: The Kalman filter is the proven optimal *linear* filter. No other filter that uses a linear combination of measurements can produce an estimate with a smaller error covariance.

Kalman Filter Optimality Conditions

The Absolute Best Estimator (A Stronger Condition)

If, in addition to the above, the noise distributions are also **Gaussian**:

$$\mathbf{w}_k \sim N(\mathbf{0}, \mathbf{Q}_k), \quad \mathbf{v}_k \sim N(\mathbf{0}, \mathbf{R}_k)$$

Conclusion: The Kalman filter is the best possible estimator of *any kind*, linear or nonlinear. It is the true minimum variance unbiased estimator.

Important Notes

- If noise is correlated (i.e., "colored"), the standard filter is no longer optimal.
- If the system is nonlinear, other variations of KF are needed to approximate the optimal solution.

The Innovations: A Health Check for Your Filter

What are the Innovations?

The innovation term represents the "new information" gained at each step:

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k \underbrace{(\mathbf{y}_k - \mathbf{H}_k \hat{\mathbf{x}}_k^-)}_{\text{Innovation: } \tilde{\mathbf{y}}_k}$$

Theoretical Properties of an Optimal Filter

If the Kalman filter is working perfectly (i.e., its model and noise parameters $\mathbf{F}, \mathbf{H}, \mathbf{Q}, \mathbf{R}$ are correct), the sequence of innovations over time should be zero-mean, white noise:

- **Zero-Mean:** $E[\tilde{\mathbf{y}}_k] = \mathbf{0}$
- **Covariance:** $\text{cov}(\tilde{\mathbf{y}}_k) = \mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k$

The Innovations: A Health Check for Your Filter

Kalman Filter Whitens the Measurement

- **Zero-Mean:** $E[\tilde{\mathbf{y}}_k] = \mathbf{0}$
- **Covariance:** $\text{cov}(\tilde{\mathbf{y}}_k) = \mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k$

Filter Diagnostics

If its statistics do not match the theory (e.g., its mean is not zero), it signals a problem with the filter.
This indicates that either the **system model** or the assumed **noise statistics** are incorrect.

The Art of Modeling

The Filter is Only as Good as its Model

The Engineer's Task

The Kalman filter is a mathematical framework. Its performance depends entirely on how accurately we model our system. This is where the engineering becomes an art.

Modeling the Dynamics

- **State Transition (F)**
- **Control Input (G)**
- **Measurement (H)**

Art of Tuning

- **Process Noise (Q)**
- **Measurement Noise (R)**
- **Initial Conditions ($\mathbf{x}_0, \mathbf{P}_0$)**

Part 1: Modeling the System Dynamics

These matrices are typically derived from the physics of the system.

- **State Transition (F):** How does the state evolve on its own? (e.g., laws of motion) This is where the core logic involves. Mostly it is physics.
- **Control Input (G):** How do known external forces affect the state? This is generally more challenging in the control systems.
- **Measurement (H):** How do our sensors relate to the state? Direct measure is easy, coupled measurement is more challenging.

The Art of Modeling

Tuning

Part 2: The "Art" of Tuning

These parameters quantify uncertainty and are often found through experimentation.

- **Process Noise (Q):** How much do we trust our model? (Accounts for unmodeled forces like friction, wind). Better **F**, **G**, and **H** leads lower **Q**. But you never be sure.
- **Measurement Noise (R):** How much do we trust our sensors? This is entirely depends on the sensor. (Often from datasheets or can be approximated).
- **Initial Conditions ($\mathbf{x}_0, \mathbf{P}_0$):** Where does the system start, and how certain are we? Generally we have bad mean and high covariance.

Continuous Time Kalman Filter

The Kalman Filter we discussed is only the discrete-time version. While most of the sensors and computational systems we use are digital, so discrete, the continuous time Kalman Filter is also available.

A Brief History of the Kalman Filter

- The **continuous-time** filter was first developed in the late 1950s by James Follin, A. G. Carlton, James Hanson, and Richard Bucy in unpublished work at the Johns Hopkins Applied Physics Lab.
- Rudolph Kalman independently developed the **discrete-time** Kalman filter in 1960.
- After becoming aware of each other's work, Kalman and Bucy collaborated on the formal publication of the continuous-time version. This is why it is often referred to as the **Kalman-Bucy filter** (AE 527).

The Continuous-Time Kalman Filter

The Kalman-Bucy Filter (AE 527)

1. The Continuous-Time System Model

$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{w}$ (State Equation)
$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{v}$ (Measurement Equation)

2. The Continuous-Time Filter Equations

Initialization: $\hat{\mathbf{x}}(0) = E[\mathbf{x}(0)], \quad \mathbf{P}(0) = E\left[(\mathbf{x}(0) - \hat{\mathbf{x}}(0))(\mathbf{x}(0) - \hat{\mathbf{x}}(0))^T\right]$
Filter Equations (for t 0):

- *Kalman Gain:* $\mathbf{K}(t) = \mathbf{P}(t)\mathbf{C}^T\mathbf{R}_c^{-1}$
- *State Estimate Evolution:* $\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{K}(t)(\mathbf{y} - \mathbf{C}\hat{\mathbf{x}})$
- *Covariance Evolution (The Riccati Equation):* $\dot{\mathbf{P}} = \mathbf{A}\mathbf{P} + \mathbf{P}\mathbf{A}^T - \mathbf{P}\mathbf{C}^T\mathbf{R}_c^{-1}\mathbf{C}\mathbf{P} + \mathbf{Q}_c$

So, What is Kalman Filter ?

The System Model

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}, \quad \mathbf{w}_k \sim N(\mathbf{0}, \mathbf{Q}_k)$$
$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k, \quad \mathbf{v}_k \sim N(\mathbf{0}, \mathbf{R}_k)$$

We use it for...

- Represent a dynamic model under noise $\rightarrow$ causes error (drift) over time.
- Represent the uncertainty and track over time.
- Correct the error made in the prediction step by noisy measurements.
- Intrinsically, a sensor fusion framework under noise.

"A robot that carries a notion of its own uncertainty and that acts accordingly is superior to one that does not." *Probabilistic Robotics Book*

Congratulations!

You have now derived the complete set of equations for the **Linear Kalman Filter**.

What's Covered

- Predict-Update Cycle (Propagation+RLS)
- *A priori* Prediction ($\hat{\mathbf{x}}_k^-$)
- *A posteriori* Estimation ($\hat{\mathbf{x}}_k^+$)
- Why Kalman Filter is Best Linear Unbiased Estimator (**BLUE**) ?
- Health Check for Filter
- Art of System Design

This was all!

Now, you know what discrete-time linear Kalman Filter is!

Code Review 2: Linear Kalman Filter

State Vector

Simple 1D system. The state is position (p_k) and velocity (v_k): $\mathbf{x}_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix}$

State Transition Model

$p_k = p_{k-1} + v_{k-1} \cdot \Delta t, \quad v_k = v_{k-1}$
This gives the state transition matrix:

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

$\mathbf{w}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ (e.g., random accelerations).

Measurement Model

The measurement matrix $\mathbf{H}$ extracts the position component from the state vector:

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

$v_k \sim \mathcal{N}(0, R)$ sensor inaccuracy.

Example: Kalman Filter in Control Systems

Problem Definition

1D Cart Control (without Pole)

Control a cart on a 1D track. Apply acceleration (a_k) but we only measure its position (p_k).

State-Space Representation

- **State Equation:**

$$x_{k+1} = Ax_k + Bu_k + w_k$$

- **Measurement Equation:**

$$y_k = Hx_k + v_k$$

State Vector (x_k)

$$x_k = \begin{bmatrix} p_k \\ \dot{p}_k \end{bmatrix}$$

Control Effort (u_k)

Our control input is the acceleration.

$$u_k = \text{acceleration}$$

Example: Kalman Filter in Control Systems

System Dynamics & Discretization

1. Kinematic Equations (Your Domain Knowledge)

$$p(t_0 + \Delta t) = p(t_0) + \dot{p}(t_0)\Delta t + \frac{1}{2}a(t_0)\Delta t^2$$

$$\dot{p}(t_0 + \Delta t) = \dot{p}(t_0) + a(t_0)\Delta t$$

2. Discretization

$t_0 = k$ and $t_0 + \Delta t = k + 1$. u_k is constant over the interval Δt.

$$p_{k+1} = p_k + \Delta t \dot{p}_k + \frac{\Delta t^2}{2} u_k$$

$$\dot{p}_{k+1} = \dot{p}_k + \Delta t u_k$$

3. State-Space Form

$$\begin{bmatrix} p_{k+1} \\ \dot{p}_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}}_{A} \begin{bmatrix} p_k \\ \dot{p}_k \end{bmatrix} + \underbrace{\begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix}}_{B} u_k \quad y_k = \underbrace{\begin{bmatrix} 1, 0 \end{bmatrix}}_{H} \begin{bmatrix} p_k \\ \dot{p}_k \end{bmatrix}$$

For our example, we set the sampling period $\Delta t = 1$ second.

Example: Kalman Filter in Control Systems

State Feedback Controller Design

State Feedback Control

Fundamental and common way of calculating the control input u_k based on the system's estimated state $\hat{x}_k$.

1. The Control Law

$$u_k = -K\hat{x}_k$$

where K is the **state feedback gain matrix**.

2. Pole Placement Method (or LQR) (Your Control Knowledge)

- Desired stable poles: $\lambda_1 = 0.5$ and $\lambda_2 = 0.7$.
- Characteristic polynomial: $\det(\lambda I - (A - BK)) = (\lambda - 0.5)(\lambda - 0.7)$

$$= \lambda^2 - 1.2\lambda + 0.35$$

3. Final Gain Matrix K

$$K = \begin{bmatrix} 0.28 & 0.6 \end{bmatrix}$$

Control gain matrix ensures driven to target quickly and without overshoot.

Example: Kalman Filter in Control Systems

The Kalman Filter: The State Estimator

1. Predict Step (Time Update)

$$\hat{x}_k^- = A\hat{x}_{k-1}^+ + Bu_{k-1} \quad \& \quad P_k^- = A_{k-1}P_{k-1}^+A_{k-1}^T + Q_{k-1}$$

2. Update Step (Measurement Update)

$$L_k = P_k^- H^T (HP_k^- H^T + R_k)^{-1} \quad (L_k\text{: Kalman Gain})$$

$$\hat{x}_k^+ = \hat{x}_k^- + L_k(y_k - H\hat{x}_k^-)$$

$$P_k^+ = (I - L_k H)P_k^-(I - L_k H)^T + K_k R_k K_k^T$$

3. Control Step

$$u_k = -K\hat{x}_k^+$$

Example: Kalman Filter in Orbit

The filter fuses high-frequency gyro data with accurate star sensor data to produce a smooth, drift-free estimate of the satellite's orientation.

State Vector ($\mathbf{x}_k$)	Input Vector ($\mathbf{u}_k$)	Measurement ($\mathbf{z}_k$)
What We Estimate	**Drives Prediction**	**Used for Correction**
3D Orientation (Quaternion)	Angular Velocity (from Rate Gyros)	Absolute Orientation (from Sun/Star Sensor)

Bonus: Kalman Filter in Wall Street

Pairs Trading

This is a classic application where you trade two historically correlated assets (e.g., Coke vs. Pepsi). The Kalman filter is used to:

- **Dynamically estimate the hedge ratio** (beta) between the two assets as it changes over time.
- **Model the spread** between them and identify statistically significant deviations, signaling when to enter a mean-reversion trade.

Estimating "True Price"

For a single asset, the filter can be used to estimate its underlying value by separating the "signal" from the "noise."

- The unobservable **"true price"** is treated as the hidden state.
- The daily **observed market price** is used as a noisy measurement.
- The filter produces a smoothed price estimate that is less volatile than the raw market data.

Bonus: Probabilistic Interpretation of Kalman Filter

Prediction

$$\overline{bel}(x_t) = \int \underbrace{p(x_t|x_{t-1}, u_t)}_{\sim\mathcal{N}(x_t; A_t x_{t-1}+B_t u_t, R_t)} \underbrace{bel(x_{t-1})}_{\sim\mathcal{N}(x_{t-1}; \mu_{t-1}, \Sigma_{t-1})} dx_{t-1}$$

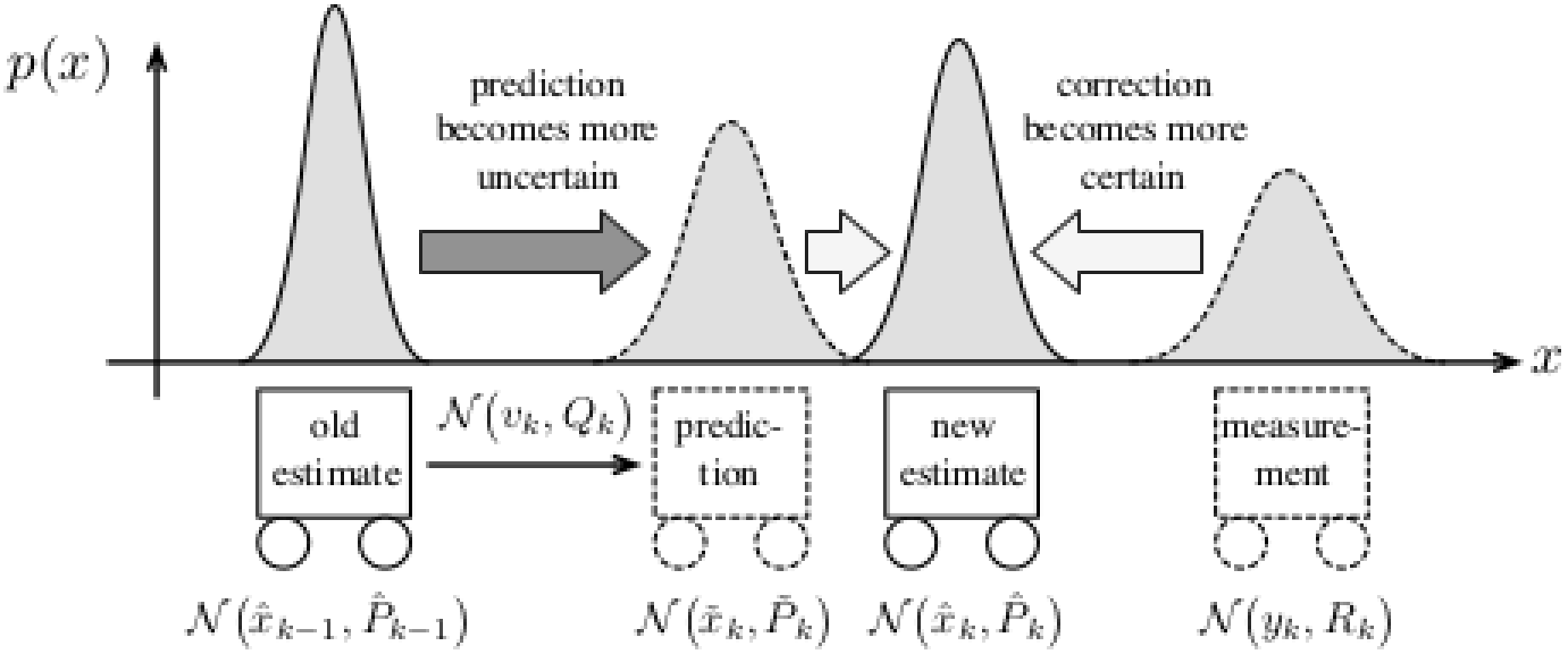
Update

$$bel(x_t) = \eta \underbrace{p(z_t|x_t)}_{\sim\mathcal{N}(y_t; C_t x_t, Q_t)} \underbrace{\overline{bel}(x_t)}_{\sim\mathcal{N}(x_t; \overline{\mu}_t, \overline{\Sigma}_t)}$$

Gaussian Assumption

When the Kalman filter is derived this way, then no conclusions can be drawn about the optimality of the filter when the noise is not Gaussian.

Probabilistic Predict-Update Cycle



Bonus: Recursive Bayesian State Estimator

1. The system and measurement equations are given as follows:

$$x_{k+1} = f_k(x_k, w_k)$$
$$y_k = h_k(x_k, v_k)$$

   where $\{w_k\}$ and $\{v_k\}$ are known pdf's.
2. Assuming that the pdf of the initial state $p(x_0)$ is known::

$$p(x_0|Y_0) = p(x_0)$$

3. For $k = 1, 2, \ldots,$ perform the following.
   (a) The *a priori* pdf is obtained:.

$$p(x_k|Y_{k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|Y_{k-1})\, dx_{k-1}$$

   (b) The *a posteriori* pdf is obtained:

$$p(x_k|Y_k) = \frac{p(y_k|x_k)p(x_k|Y_{k-1})}{\int p(y_k|x_k)p(x_k|Y_{k-1})\, dx_k}$$

Other Notations

Prediction

This stage has many names: Propagation, Control update, Time update, A priori estimate, ...

Various notations are used $\hat{x}_k^- = \bar{bel}(x_k) = \check{x}_k = \hat{x}_k(-)$

Update

This has also another names: Measurement, Correction, A posteriori estimate, ...

Various notations: $\hat{x}_k^+ = bel(x_k) = \hat{x}_k = \hat{x}_k(+)$

Chapter 4: Non-linear Kalman Filter

Linear Systems Do Not Really Exist

The standard Kalman Filter is the optimal estimator for *linear* systems. But what happens when our system is nonlinear, as most real-world systems are (e.g., robot kinematics, vehicle dynamics, weather patterns)?

Linear Algebra

Linear algebra is only useful the system of equations can be expressed as linear matrix operations.

Nonlinear Filtering

Nonlinear filtering can be a difficult and complex subject. It is certainly not as mature, cohesive, or well understood as linear filtering. However, some nonlinear estimation methods have become widespread, including nonlinear extensions of the Kalman filter.

Code Review 3

Linear Kalman Filter on 2D Nonlinear Data

- The state vector: $x_k = \left[p_{x,k}, p_{y,k}, v_{x,k}, v_{y,k}\right]^T$
- True nonlinear model:

$$
\begin{aligned}
v_{x,k} &= v_{x,k-1}\cos(\omega\Delta t) - v_{y,k-1}\sin(\omega\Delta t) \\
v_{y,k} &= v_{x,k-1}\sin(\omega\Delta t) + v_{y,k-1}\cos(\omega\Delta t) \\
p_{x,k} &= p_{x,k-1} + v_{x,k}\Delta t \\
p_{y,k} &= p_{y,k-1} + v_{y,k}\Delta t
\end{aligned}
$$

- The incorrect linear model, F_{linear}, (assumes constant velocity):

$$
F_{linear} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad v_{x,k} = v_{x,k-1} \quad \& \quad v_{y,k} = v_{y,k-1}
$$

Nonlinear Models

Two Types of Nonlinearity

Our elegant matrix equations must be replaced:

1. Nonlinear System Dynamics:

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \ldots \quad \rightarrow \quad \mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1})$$

The state no longer evolves linearly.

2. Nonlinear Measurements:

$$\mathbf{y}_k = \mathbf{H}\mathbf{x}_k + \ldots \quad \rightarrow \quad \mathbf{y}_k = h(\mathbf{x}_k + \mathbf{v}_k)$$

The sensor measurement is a nonlinear function of the state.

Beyond Linearity: The Extended Kalman Filter (EKF)

Extended Kalman Filter (EKF) handles nonlinearity by approximating the system with a linear model at each time step. It finds the best *linear fit* to the nonlinear function at the current state estimate, and then applies the familiar Kalman Filter prediction and correction steps.

The Core Strategy

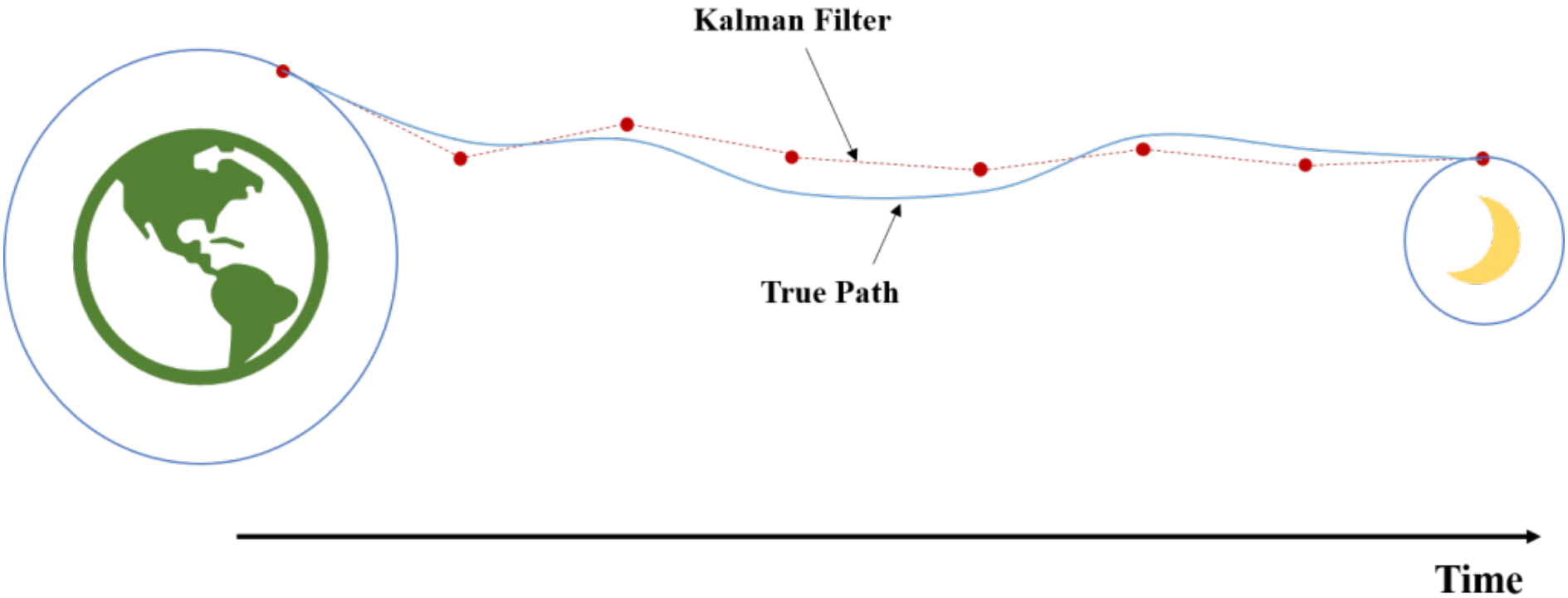
Think of it as running the standard Kalman Filter on a model that is constantly being updated to reflect the local dynamics.

Forward to the Moon!

That's one small step for man, one giant leap for mankind

Space Proven!

It was during a visit by Kálmán to the **NASA Ames Research Center** that Schmidt saw the applicability of Kálmán's ideas to the nonlinear problem of trajectory estimation for the **Apollo program** resulting in its incorporation in the Apollo navigation computer.



What is Linearization ?

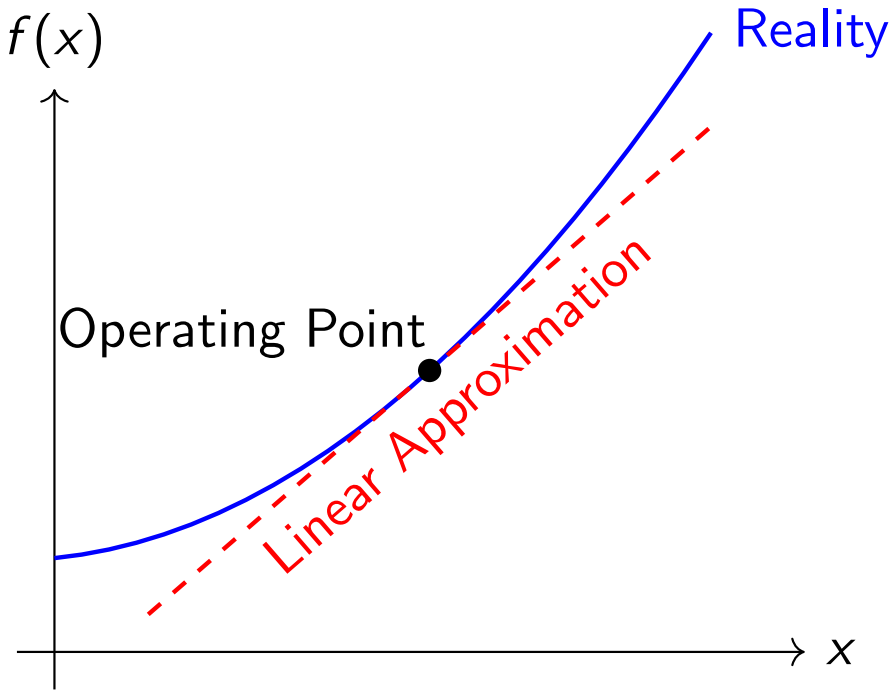


Figure: Linear models are often just a local approximation of a nonlinear reality.

Recall: Taylor Series Approximation

A nonlinear function can be approximated by a linear function around a specific point using a first-order Taylor series expansion.

First Order Taylor Series Expansion

For a vector function $f(x)$, the approximation around a point x_0 is:

$$f(x) \approx f(x_0) + \underbrace{\frac{\partial f}{\partial x}}_{\text{Jacobian}} \Bigg|_{x_0} (x - x_0) + H.O.T$$

Section 4.1: Linearization of Multivariate Functions

We will show how to **linearize** a nonlinear system.

The General Nonlinear System Model

The system dynamics and measurements are described by nonlinear functions, $f(\cdot)$ and $h(\cdot)$:

$$\mathbf{x}_k = f_{k-1}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1}) \quad \mathbf{w}_k \sim N(\mathbf{0}, \mathbf{Q}_k) \qquad \text{(Dynamics)}$$

$$\mathbf{y}_k = h_k(\mathbf{x}_k, \mathbf{v}_k) \quad \mathbf{v}_k \sim N(\mathbf{0}, \mathbf{R}_k) \qquad \text{(Measurement)}$$

Linearization of Multivariate Functions

Taylor Approximation of Dynamics

We linearize the nonlinear state equation around the most recent state estimate ($\hat{x}_{k-1}^{+}$) and assume zero process noise ($w_{k-1} = 0$).

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

$$\mathbf{x}_k \approx f_{k-1}(\hat{x}_{k-1}^{+}, u_{k-1}, 0) + \underbrace{\left.\frac{\partial f_{k-1}}{\partial x}\right|_{\hat{x}_{k-1}^{+}}}_{F_{k-1}} (x_{k-1} - \hat{x}_{k-1}^{+}) + \underbrace{\left.\frac{\partial f_{k-1}}{\partial w}\right|_{\hat{x}_{k-1}^{+}}}_{L_{k-1}} (w_{k-1} - 0)$$

$$= f_{k-1}(\hat{x}_{k-1}^{+}, u_{k-1}, 0) + F_{k-1}(x_{k-1} - \hat{x}_{k-1}^{+}) + \underbrace{L_{k-1} w_{k-1}}_{\tilde{w}_{k-1}}$$

$$= f_{k-1}(\hat{x}_{k-1}^{+}, u_{k-1}, 0) + F_{k-1}(x_{k-1} - \hat{x}_{k-1}^{+}) + \tilde{w}_{k-1}$$

Linearization of Multivariate Functions

Taylor Approximation of Dynamics cont"d

$$x_k = f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}, 0) + F_{k-1}(x_{k-1} - \hat{x}_{k-1}^+) + \tilde{w}_{k-1}$$

Derivation of $\tilde{w}_k$ Covariance

$$\tilde{w}_k \sim (0, (L_k w_k)(L_k w_k)^T)$$
$$\tilde{w}_k \sim (0, (L_k w_k w_k^T L_k^T))$$
$$\tilde{w}_k \sim (0, L_k Q_k L_k^T)$$

Linearization of Multivariate Functions

Taylor Approximation of Measurements

Similarly, we linearize the measurement equation around the *a priori* prediction $x_k = \hat{x}_k^-$ and zero measurement noise $v_k = 0$:

$$\begin{aligned}
\mathbf{y}_k &= h_k(\mathbf{x}_k, \mathbf{v}_k) \\
\mathbf{y}_k &\approx h_k(\hat{x}_k^-, 0) + \underbrace{\left.\frac{\partial h_k}{\partial x}\right|_{\hat{x}_k^-}}_{H_k}(x_k - \hat{x}_k^-) + \underbrace{\left.\frac{\partial h_k}{\partial v}\right|_{\hat{x}_k^-}}_{M_k}(v_k - 0) \\
&= h_k(\hat{x}_k^-, 0) + H_k(x_k - \hat{x}_k^-) + \underbrace{M_k v_k}_{\tilde{v}_k} \\
&= h_k(\hat{x}_k^-, 0) + H_k(x_k - \hat{x}_k^-) + \tilde{v}_k
\end{aligned}$$

Linearization of Multivariate Functions

Taylor Approximation of Measurements cont'd

$$y_k = h_k(\hat{x}_k^-, 0) + H_k(x_k - \hat{x}_k^-) + \tilde{v}_k$$

Derivation of $\tilde{v}_k$ Covariance

$$\tilde{v}_k \sim (0, (M_k v_k)(M_k v_k)^T)$$
$$\tilde{v}_k \sim (0, (M_k v_k v_k^T M_k^T))$$
$$\tilde{v}_k \sim (0, M_k R_k M_k^T)$$

Linearization of Multivariate Functions

Complete Set of Linearized System of Equations

Taylor Approximation of Nonlinear System

$$x_k = f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}, 0) + F_{k-1}(x_{k-1} - \hat{x}_{k-1}^+) + \tilde{w}_{k-1} \quad \text{(Dynamics)}$$
$$y_k = h_k(\hat{x}_k^-, 0) + H_k(x_k - \hat{x}_k^-) + \tilde{v}_k \quad \text{(Measurements)}$$

$$F_{k-1} = \left.\frac{\partial f_{k-1}}{\partial x}\right|_{\hat{x}_{k-1}^+} \qquad L_{k-1} = \left.\frac{\partial f_{k-1}}{\partial w}\right|_{\hat{x}_{k-1}^+}$$

$$H_k = \left.\frac{\partial h_k}{\partial x}\right|_{\hat{x}_k^-} \qquad M_k = \left.\frac{\partial h_k}{\partial v}\right|_{\hat{x}_k^-}$$

$$\tilde{v}_k \sim (0, M_k R_k M_k^T) \quad \tilde{w}_k \sim (0, L_k Q_k L_k^T)$$

Kalman Equations on Linearized System

Once we linearized the system, the equations we derived for linear Kalman Filter become available.

1. **Propagate (Predict) Next State:** $\hat{x}_k^- = f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}, 0)$
2. **Propagate Covariance:** $P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + \underbrace{L_{k-1} Q_{k-1} L_{k-1}^T}_{\tilde{Q}_{k-1}}$
3. **Calculate Kalman Gain:** $K_k = P_k^- H_k^T (H_k P_k^- H_k^T + \underbrace{M_k R_k M_k^T}_{\tilde{R}_k})^{-1}$
4. **Estimate State:** $\hat{x}_k^+ = \hat{x}_k^- + K_k[y_k - h_k(\hat{x}_k^-, 0)]$
5. **Estimate New Uncertainty:**
   $P_k^+ = (I - K_k H_k) P_k^- (I - K_k H_k)^T + K_k \underbrace{M_k R_k M_k^T}_{\tilde{R}_k} K_k^T$

EKF: Putting Everything Together

Nonlinear System and Initialization

1. System and Measurement Equations

$$
\begin{aligned}
x_k &= f_{k-1}(x_{k-1}, u_{k-1}, w_{k-1}) \\
y_k &= h_k(x_k, v_k) \\
w_k &\sim (0, Q_k) \\
v_k &\sim (0, R_k)
\end{aligned}
$$

2. Initialization

$$
\begin{aligned}
\hat{x}_0^+ &= E[x_0] \\
P_0^+ &= E\left[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T\right]
\end{aligned}
$$

3. EKF Loop: (for k = 1, 2, ...)

(a) Perform Propagation of State and Covariance

The system dynamics f_{k-1} is linearized around the previous state estimate $\hat{x}_{k-1}^{+}$. The state and error covariance are projected forward in time.

$$\hat{x}_k^- = f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}, 0)$$
$$P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + L_{k-1} Q_{k-1} L_{k-1}^T$$

$$F_{k-1} = \left.\frac{\partial f_{k-1}}{\partial x}\right|_{\hat{x}_{k-1}^+} \qquad L_{k-1} = \left.\frac{\partial f_{k-1}}{\partial w}\right|_{\hat{x}_{k-1}^+}$$

EKF: Putting Everything Together

Loop: Measurements

3. EKF Loop: (for k = 1, 2, ...)

(b) Perform Measurement Update of State and Covariance The predicted state is corrected using the incoming measurement y_k.

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + M_k R_k M_k^T)^{-1}$$
$$\hat{x}_k^+ = \hat{x}_k^- + K_k[y_k - h_k(\hat{x}_k^-, 0)]$$
$$P_k^+ = (I - K_k H_k) P_k^- (I - K_k H_k)^T + K_k M_k R_k M_k^T K_k^T$$

$$H_k = \left.\frac{\partial h_k}{\partial x}\right|_{\hat{x}_k^-} \qquad M_k = \left.\frac{\partial h_k}{\partial v}\right|_{\hat{x}_k^-}$$

Key Differences Between Linear vs. Extended KF

System of Equations

Linear System

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k$$

Nonlinear System

$$\mathbf{x}_k = \mathbf{f_{k-1}}(x_{k-1}, u_{k-1}, w_{k-1})$$

$$\mathbf{y}_k = \mathbf{h_k}(x_k, v_k)$$

Key Differences Between Linear vs. Extended KF

Linearized System

F_{k-1} and H_k varies by time but does not depend on estimated state $\hat{x}$:

Linear Model

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k$$

F_{k-1} and H_k depends on estimated state $\hat{x}$ implicitly (due to Jacobian):

Nonlinear Model

$$x_k = f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}, 0) + F_{k-1}(x_{k-1} - \hat{x}_{k-1}) + \tilde{w}_{k-1}$$

$$y_k = h_k(\hat{x}_k^-, 0) + H_k(x_k - \hat{x}_k^-) + \tilde{v}_k$$

$$F_{k-1} = \left.\frac{\partial f_{k-1}}{\partial x}\right|_{\hat{x}_{k-1}^+} \qquad H_k = \left.\frac{\partial h_k}{\partial x}\right|_{\hat{x}_k^-}$$

Simplification for Additive Noise

In many real-world systems, the process and measurement noise are **additive**. This means the noise is simply added to the state or measurement, rather than being part of a more complex function.

State Model: $x_k = f_{k-1}(x_{k-1}, u_{k-1}) + w_{k-1}$

Measurement Model: $y_k = h_k(x_k) + v_k$

This assumption greatly simplifies the Jacobians L_{k-1} and M_k:

- **Process Noise Jacobian (L_{k-1})**

$$L_{k-1} = \frac{\partial f_{k-1}}{\partial w_{k-1}} = \frac{\partial}{\partial w_{k-1}}(f_{k-1} + w_{k-1}) = 0 + I = I$$

- **Measurement Noise Jacobian (M_k)**

$$M_k = \frac{\partial h_k}{\partial v_k} = \frac{\partial}{\partial v_k}(h_k + v_k) = 0 + I = I$$

Simplification for Additive Noise

Therefore our equations for Kalman Gain and Covariances in EKF reduces to Linear Kalman Filter equations:

EKF with Additive Noise

1. **Propagate (Predict) Next State:** $\hat{x}_k^- = f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}, 0)$
2. **Propagate Covariance:** $P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + \underbrace{I Q_{k-1} I^T}_{Q_{k-1}}$
3. **Calculate Kalman Gain:** $K_k = P_k^- H_k^T (H_k P_k^- H_k^T + \underbrace{I R_k I^T}_{R_k})^{-1}$
4. **Estimate State:** $\hat{x}_k^+ = \hat{x}_k^- + K_k[y_k - h_k(\hat{x}_k^-, 0)]$
5. **Estimate New Uncertainty:**
   $P_k^+ = (I - K_k H_k) P_k^- (I - K_k H_k)^T + K_k \underbrace{I R_k I^T}_{R_k} K_k^T$

Section 4.3: Limitations of the EKF

Linearization Error

The EKF is prone to linearization error when:

- The system dynamics are highly nonlinear.
- The sensor sampling time is slow relative to how fast the system is evolving.

This has two important consequences:

- The estimated mean state can become very different from the true state.
- The estimated state covariance can fail to capture the true uncertainty in the state.

Key Consequence

Linearization error can cause the estimator to be **overconfident in a wrong answer!**

Limitations of the EKF

Computing Jacobians

Computing Jacobian matrices for complicated nonlinear functions is also a common source of error in EKF implementations!

- Analytical differentiation is prone to human error.
- Numerical differentiation can be slow and unstable.
- Automatic differentiation (e.g., at compile time) can also behave unpredictably.

Differentiability is NOT guaranteed!

What if one or more of our models is non-differentiable?
Do we really need linearization for nonlinear Kalman filtering?

Congratulations!

You have now explored the theory, derivation, and potential pitfalls of the the most widely applied state estimation algorithm for nonlinear systems, **Extended Kalman Filter**.

What You've Learned

- Why the standard Kalman Filter isn't enough for **nonlinear systems**.
- The core EKF strategy: **Linearize-and-Apply**.
- How to compute the **Jacobian matrices** (F, H, L, M) that form the linear approximation.
- The complete **EKF Predict-Correct cycle**.
- The common simplification for **additive noise** $(L = I, M = I)$.
- The key **limitations of the EKF**, including linearization errors and challenges with Jacobians.

Industry Standard

This may even secure you a job!

Code Review 4

2D Vehicle Navigation

- Our state includes the vehicle's position, speed, heading, and turn rate:

$$\mathbf{x}_k = \begin{bmatrix} p_{x,k} & p_{y,k} & v_k & \theta_k & \omega_k \end{bmatrix}^T$$

- We assume speed (v) and turn rate (ω) are **constant** over the small time step Δt. The heading is then predicted as:

$$\theta_k = \theta_{k-1} + \omega_{k-1}\Delta t$$

  The position update is derived by integrating this changing heading.

- The EKF cycle starts with this prediction, which is then corrected using incoming **position measurements**.

Section 4.4: Why Kalman Filters are a Robotics Superpower

1. Foundation for Control

Effective control requires accurate state knowledge. Kalman filters provide a robust estimate, which is essential for any closed-loop control law. You can't control what you don't know.

2. Powerful Sensor Fusion

They provide a rigorous framework to fuse data from multiple sensors arriving at different rates (*asynchronously*). This allows a robot to combine the strengths of IMUs, GPS, cameras, and LiDAR for a single, superior state estimate.

Why Kalman Filters are a Robotics Superpower

3. Principled Uncertainty Management

The filter doesn't just tell you *where* the robot is; it tells you *how sure* it is via the covariance matrix ($\mathbf{P}$). This is crucial for safety-critical decisions, path planning, and robust operation in the real world.

4. Predictive Power & Robustness

The filter's prediction step allows it to estimate the state ahead in time, compensating for sensor lag and enabling smooth operation even with temporary data loss (e.g., GPS outage in a tunnel or camera occlusion).

Section 4.5: EKF for Rover Sensor Fusion: IMU

1. Prediction Step (Propagation)

Driven by an Inertial Measurement Unit (IMU):

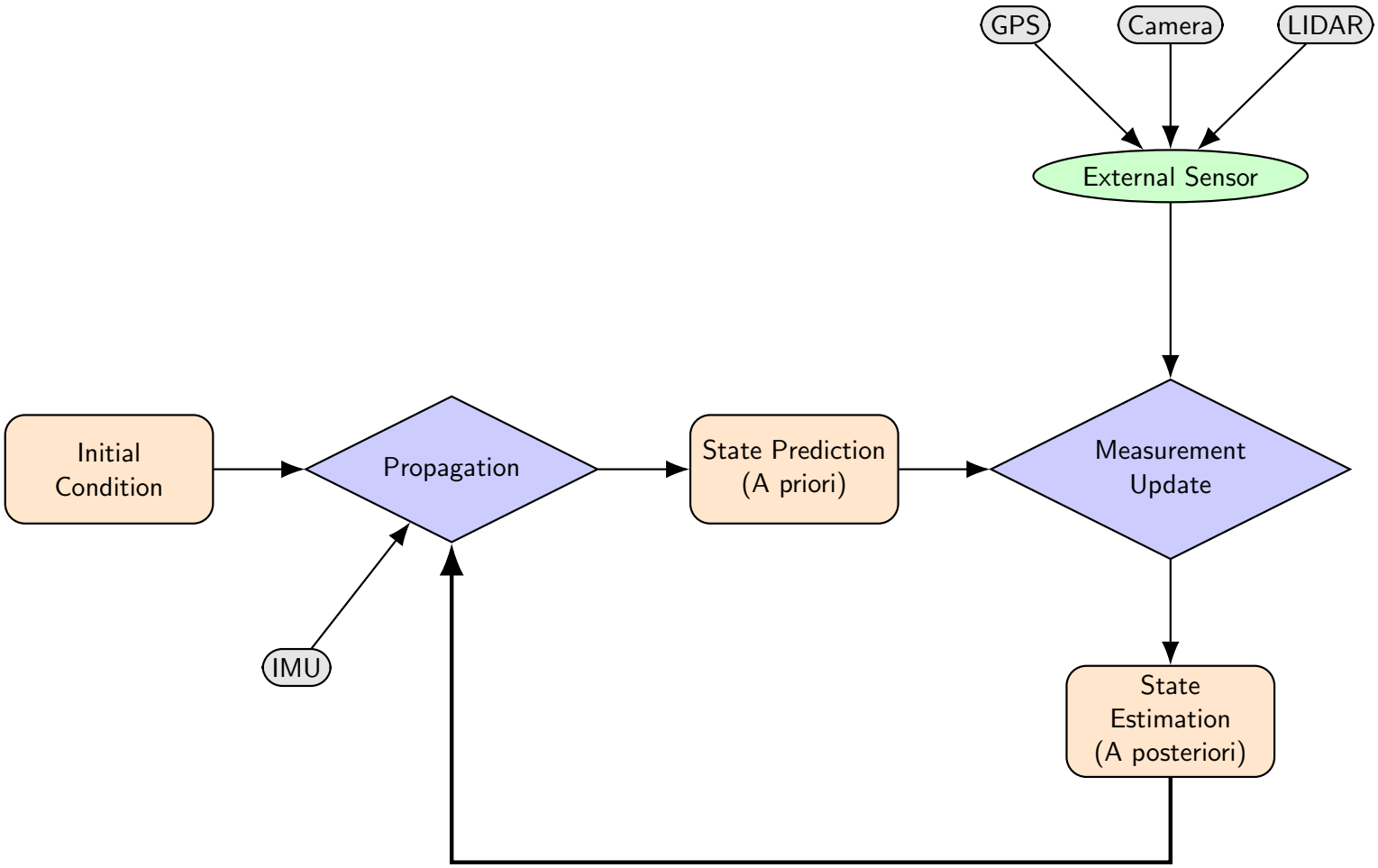
- Provides high-frequency acceleration and angular velocity.
- This data is treated as a **control input ($\mathbf{u}_k$)** that drives the state prediction forward.
- The state transition model becomes an *Inertial Navigation System (INS) mechanization*, using physics to dead-reckon the robot's state between corrections.

2. Correction Step (Update)

Anchored by external measurements:

- **GPS:** Provides a direct, absolute measurement of position (p_x, p_y) and sometimes velocity. It's a simple and powerful way to correct for the IMU's accumulated drift.
- **Perception Sensors:** Vision, LiDAR, and RADAR provide rich information about the robot's motion relative to its environment.

Conceptual Flow of a Sensor Fusion Filter



Code Review 5

Rover Navigation with IMU & GPS

The Core Idea: Sensor Fusion

- **IMU (High-Frequency, 400 Hz):** Provides acceleration and angular velocity data. We use it for **propagation** step. This is often called "dead reckoning".
- **GPS (Low-Frequency, $\sim$ 1 Hz):** Provides accurate but infrequent absolute position data. We use this for the **update** step to eliminate the drift that accumulates from integrating the IMU data.

The State Vector (x)

The state we are estimating is:

$$\mathbf{x} = \begin{bmatrix} p_x \\ p_y \\ \theta \\ v \end{bmatrix}$$

IMU as Control Input (u)

The IMU readings are not part of the state, but act as the control inputs that drive the state forward:

$$\mathbf{u}_k = \begin{bmatrix} a_k \\ \omega_k \end{bmatrix}$$

IMU Role: Propagation vs. Measurement

Wheeled Rovers: State Propagation (Simple, predictable motion)

The IMU's acceleration ($\mathbf{a}$) and angular velocity (ω) are integrated to directly predict the next state. This is called **dead reckoning**.

$$\text{IMU data } (\mathbf{a}, \omega) \xrightarrow{\int dt} \text{ State Prediction } (x_k^-)$$

The IMU **drives the prediction**; other sensors (GPS, encoders) provide corrections.

Legged Robots: State Measurement: Complex, dynamic motion

The IMU data is a critical **measurement** used to correct a prediction made by a separate physics model.

$$\text{IMU data } (\mathbf{a}, \omega) \rightarrow \text{ State Correction } (x_k^- \rightarrow x_k^+)$$

The IMU **corrects the prediction**; it cannot be integrated directly due to complex forces (e.g., foot impacts).

Example: Quadruped State: IMU as Measurement

State Vector: $x_k = \left[\mathbf{p}_{body}, \mathbf{v}_{body}, \mathbf{q}_{joints}, \mathbf{w}_{joints},\right]^T \in \mathbb{R}^{36}$

- $\mathbf{p}_{body}$ (6D): Position (x, y, z) and orientation (roll, pitch, yaw) of the torso.
- $\mathbf{v}_{body}$ (6D): Linear and angular velocity of the torso.
- $\mathbf{q}_{joints}$ (12D): The angles of each joint (e.g., 3 joints per leg x 4 legs).
- $\mathbf{w}_{joints}$ (12D): The angular velocity of each joint.

Measurement Vector: $y_k = \left[\mathbf{y}_{IMU}, \mathbf{y}_{enc}, \mathbf{y}_{contact}\right]^T \in \mathbb{R}^{22}$

- $\mathbf{y}_{IMU}$ (6D): Linear acceleration and angular velocity.
- $\mathbf{y}_{enc}$ (12D): Angle of 12 joints.
- $\mathbf{y}_{contact}$ (4D): A binary value of 4 feet for ground contact.

Control Input: $u_k = \tau \in \mathbb{R}^{12}$

τ (12D): The torque applied to each of the 12 leg joints.

The Measurement Function (H): A Sensor Model

The Problem: H is Not a Selector

When a complex model of INS utilized, the 'H' function doesn't just select states. It must model the sensor's physics to predict the expected measurement based on the current state.

Example: IMU on a Quadruped (Coupled Measurement)

The accelerometer reading is not the robot's world acceleration; it's also affected by the robot's orientation with respect to gravity.
The measurement function $h(x_k)$ that relates state to measurement is:

$$\underbrace{\mathbf{a}_{imu}}_{\text{Expected Measurement}} = \underbrace{\mathbf{R}^T}_{\text{From State}} \left(\underbrace{\mathbf{a}_{world}}_{\text{From State}} - \underbrace{\mathbf{g}}_{\text{Constant}} \right)$$

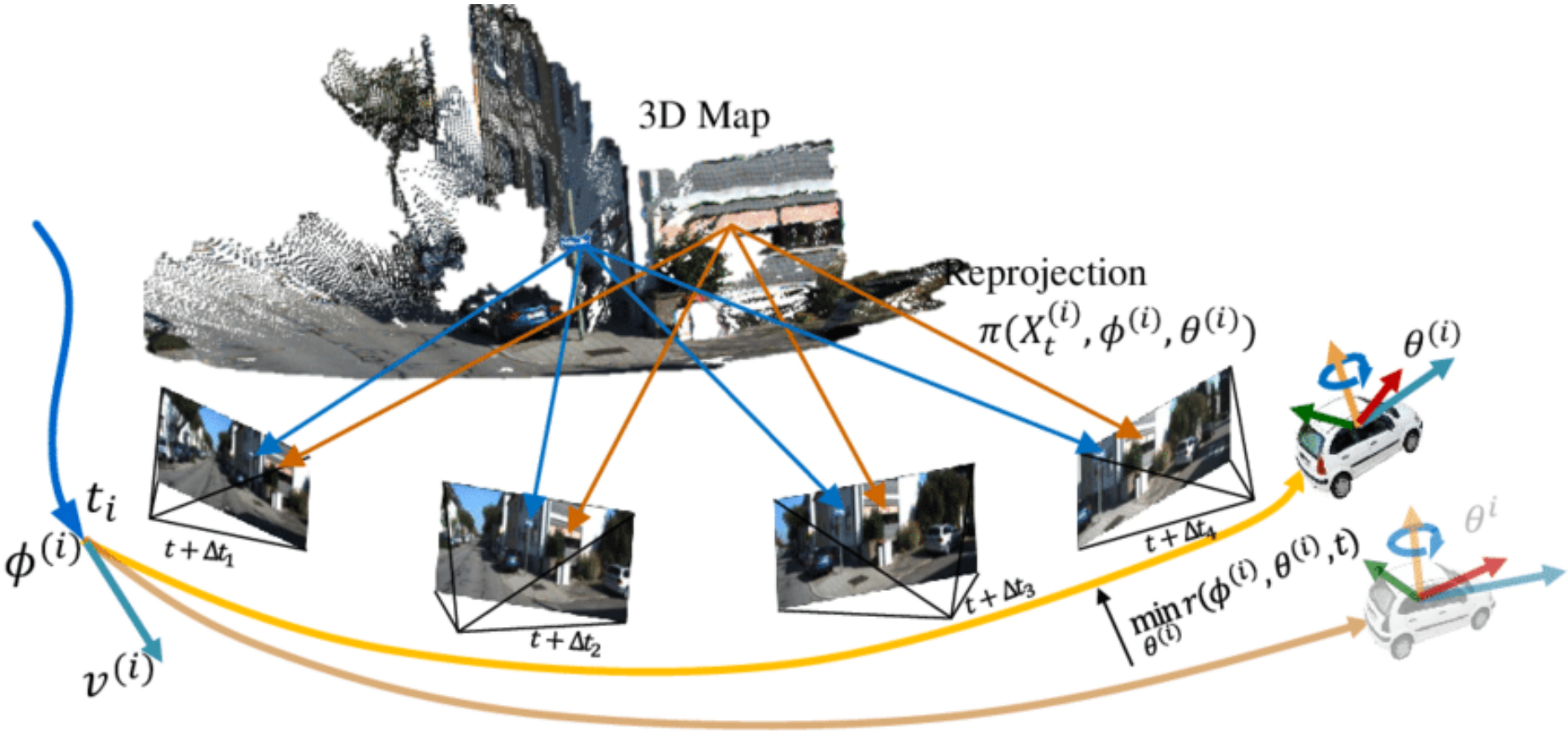
Consequence: Non-Linearity

This relationship is **non-linear** due to the rotation matrix $\mathbf{R}$. Therefore, we use an EKF.

Section 4.6: Integrating Perception

The Case of Visual Odometry

Sensors like cameras don't measure state directly. Their data must be processed by a separate algorithm first.



Integrating Perception: The Case of Visual Odometry

The Visual Odometry Workflow

1. **Computer Vision algorithm** processes images to calculate the robot's *relative* change in position and orientation.
2. This output is then passed to the EKF as a **measurement vector ($\mathbf{y}_k$)** (correction).
3. A robust VO algorithm **must** also provide a **covariance matrix ($\mathbf{R}_k$)**.

Measurement Model Complexity

The measurement matrix ($\mathbf{H}_k$) maps the filter's state to the VO measurement.

- This can be simple if VO provides direct state measurements (e.g., velocity).
- It can become very complex if the measurement is a coupled variable that relates to multiple states simultaneously.

Visual-Inertial Odometry (VIO) Pipeline

Goal: Estimate Scaled Motion

To solve the scale ambiguity of a single camera (monocular VO), we fuse its data with an IMU, which provides a true sense of scale.

Inputs

- Camera Images
- IMU Data $(\mathbf{a}, \omega)$

Output

- Relative Pose Estimate $(\Delta\mathbf{p}, \Delta\mathbf{R})$

Camera Images $\rightarrow$ *Feature Tracking* $\rightarrow$ **Scaleless Motion**

IMU Data $\rightarrow$ *Integration* $\rightarrow$ **Scaled, Drifty Motion**

$\Downarrow$

VIO Fusion (Filter/Optimization)

$\Downarrow$

Final Output: Scaled Relative Pose

VIO Integration with a Kalman Filter

Loosely-Coupled Approach

- The VIO system acts like a "black-box" motion sensor.
- It provides a complete **relative pose estimate** to the Kalman Filter.
- The KF's measurement function is simpler, as it just compares its predicted pose change to the one from the VIO.
- **Pro:** Simpler to implement.
- **Con:** Less optimal as information is lost.

Tightly-Coupled Approach

- There is no separate VIO system; the fusion happens inside the main KF.
- **Raw feature positions** from the images are used directly as measurements.
- The 'H' function becomes very complex, handling the full transformation from 3D world state to 2D image features.
- **Pro:** More accurate and robust.
- **Con:** Highly complex and computationally expensive.

VIO: Quantifying Uncertainty (Covariance)

1. Reprojection Error

Project the 3D points back into the new image. The distance between the reprojected point and the actual detected feature is the reprojection error.

- **Large reprojection error $\rightarrow$ High uncertainty $\rightarrow$ Larger covariance**

2. Number and Quality of Features

Quantity and geometric distribution of the features being tracked.

- **Fewer features or poor distribution $\rightarrow$ High uncertainty $\rightarrow$ Larger covariance**

Other options include **3D Geometric Consistency**, and **Scene Content** like illumination, blur, and repetitive texture (common in space).

EKF-SLAM (Ready-to-use in Matlab)

Core Idea: Estimate a large state vector containing the robot's pose **and** the 3D positions of all landmarks in the map simultaneously.

The State Vector The state grows as new landmarks are discovered.

$$x = \left[\begin{array}{c} x_v \\ \hline p_1 \\ p_2 \\ \vdots \\ p_N \end{array}\right]$$

The Covariance Matrix

$$P = \begin{bmatrix} \mathbf{P}_{vv} & \mathbf{P}_{vm} \\ \mathbf{P}_{mv} & \mathbf{P}_{mm} \end{bmatrix}$$

$\mathbf{P}_{vv}$ Vehicle's uncertainty.

$\mathbf{P}_{mm}$ Map's uncertainty.

$\mathbf{P}_{vm}$ **Vehicle-Map Correlations.**

When a known landmark is observed, the uncertainty of both the vehicle *and* correlated parts of the map is reduced. (Too optimistic in practice.)

The Challenge: Computational Complexity

The size of the covariance matrix grows quadratically with the number of landmarks (N). The cost of updating it is $O(N^2)$ (impractical).

Chapter 5: Variations of Kalman Filter

Kalman Filter, especially EKF, has many many different variations depending on the specific needs of the application.
Popular ones include but not limited to:

- Error State
- MSCKF
- UKF
- Particle Filtering (Not a KF)

Section 5.1: Error-State Kalman Filter

The Idea

Instead of estimating a large, non-linear state directly, we split the problem. We use a simple non-linear model for a **nominal state** and a Kalman Filter for its small **error state**.

The states are related by a composition ($\oplus$), generally simple addition.

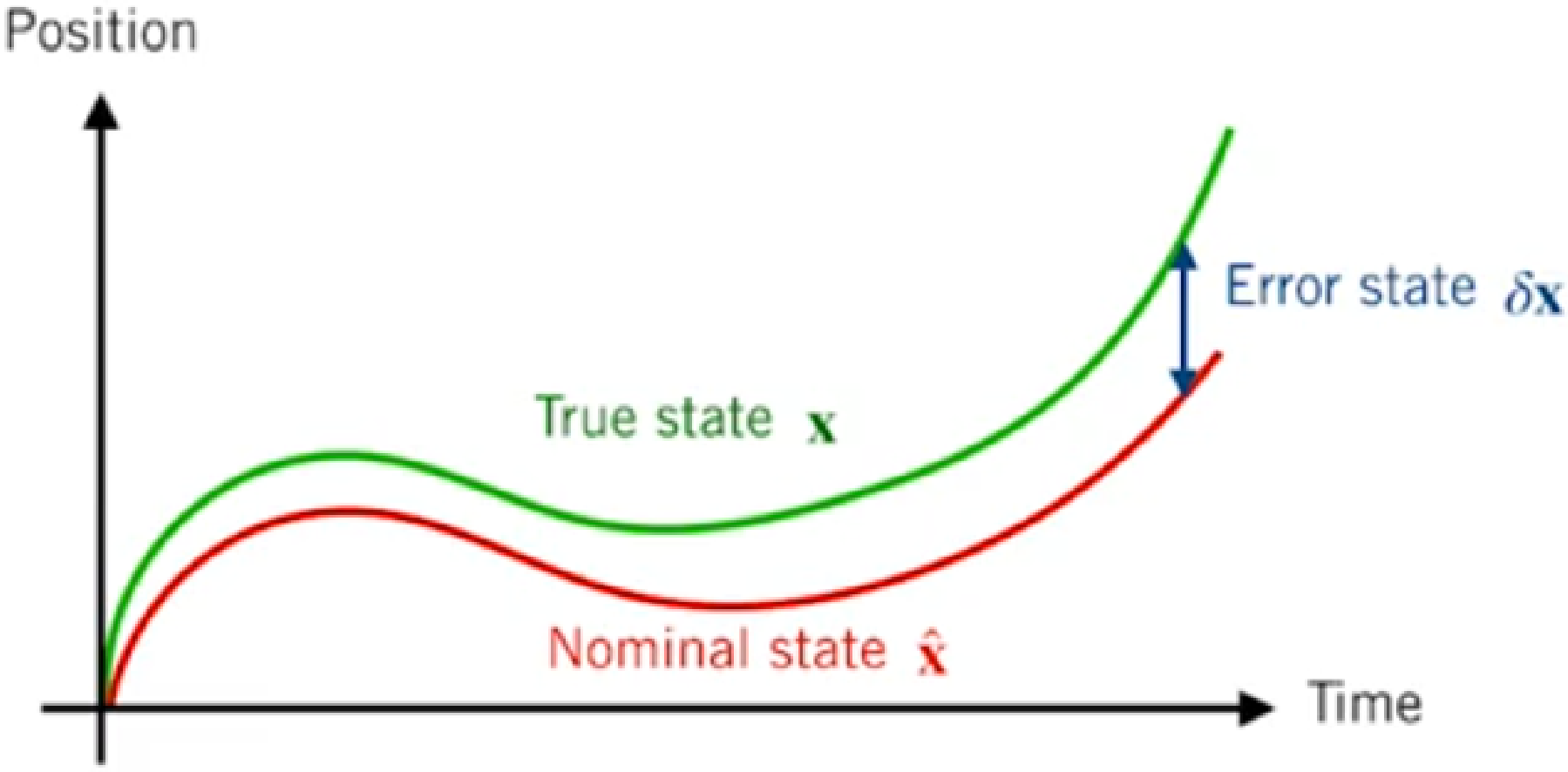
$$x = \hat{x} \oplus \delta x$$

- True State (x) :Reality we don't know
- Nominal State ($\hat{x}$): Our best guess
- Error State (δx): Our deviation which we don't know either

Sometimes referred as **Indirect** Kalman Filter.

Error State Visualization

$$x = \hat{x} \oplus \delta x$$



Replace True State with Nominal $\oplus$ Error

Dynamics

$$x_k = f_{k-1}(x_{k-1}, u_{k-1}, w_{k-1})$$

$$x_k = \hat{x} \oplus \delta x = f_{k-1}(\hat{x}, \delta x, u_{k-1}, w_{k-1})$$

Measurement

$$y_k = h(\hat{x}, \delta x, v_k)$$

Taylor Approximation

Let's assume $\oplus$ is simple addition $(+)$.

Dynamics

$$\hat{x} \oplus \delta x \approx f_{k-1}(\hat{x}_{k-1}^{+}, \delta\hat{x}_{k-1}^{+}, u_{k-1}, 0) + F_{k-1}(\delta x_{k-1} - \delta\hat{x}_{k-1}^{+}) + \tilde{w}_{k-1}$$

Measurement

$$y_k \approx h(\hat{x}_k^{-}, \delta\hat{x}_k^{-}, 0) + H_k(\delta x_k - \delta\hat{x}_k^{-}) + \tilde{v}_k$$

Main difference is:

$$F_{k-1} = \left.\frac{\partial f_{k-1}}{\partial \delta x}\right|_{\delta\hat{x}_{k-1}^{+}} \qquad L_{k-1} = \left.\frac{\partial f_{k-1}}{\partial w}\right|_{\delta\hat{x}_{k-1}^{+}}$$

$$H_k = \left.\frac{\partial h_k}{\partial \delta x}\right|_{\delta\hat{x}_k^{-}} \qquad M_k = \left.\frac{\partial h_k}{\partial v}\right|_{\delta\hat{x}_k^{-}}$$

ESKF: Unbiased Error

We consider no error $\delta\hat{x} = 0$ until we receive a measurement. Also, $\hat{x} = f_{k-1}(\hat{x}^+_{k-1}, 0, u_{k-1}, 0)$ by definition.

Dynamics and Measurement

$$\begin{aligned}
\hat{x} \oplus \delta x &\approx f_{k-1}(\hat{x}^+_{k-1}, \delta\hat{x}^+_{k-1}, u_{k-1}, 0) + F_{k-1}(\delta x_{k-1} - \delta\hat{x}^+_{k-1}) + \tilde{w}_{k-1} \\
&= F_{k-1}\delta x_{k-1} + f_{k-1}(\hat{x}^+_{k-1}, \delta\hat{x}^+_{k-1}, u_{k-1}, 0) - F_{k-1}\delta\hat{x}^+_{k-1} + \tilde{w}_{k-1} \\
&= F_{k-1}\delta x_{k-1} + f_{k-1}(\hat{x}^+_{k-1}, 0, u_{k-1}, 0) + \tilde{w}_{k-1} \\
E[\delta x] &= F_{k-1}E[\delta x_{k-1}] + E[\tilde{w}_{k-1}] \\
E[\delta x] &= F_{k-1}E[\delta x_{k-1}]
\end{aligned}$$

Similarly:

$$\begin{aligned}
y_k &\approx H_k\delta x_k + h(\hat{x}^-_k, \delta\hat{x}^-_k, 0) - H_k\delta\hat{x}^-_k + \tilde{v}_k \\
y_k &= H_k\delta x_k + h(\hat{x}^-_k, 0, 0) + \tilde{v}_k \\
0 &= H_k E[\delta x_k] + E[\tilde{v}_k]
\end{aligned}$$

ESKF Algorithm: Prediction

Prediction

$$\delta\hat{x}_k^- = F_{k-1}\delta\hat{x}_{k-1}^+ = 0 \leftarrow \text{Always } 0$$
$$\hat{x}_k^- = f_{k-1}(\hat{x}_{k-1}^+, 0, u_{k-1}, 0)$$
$$P_k^- = F_{k-1}P_{k-1}^+F_{k-1} + \tilde{Q}_{k-1}$$

ESKF Algorithm: Update

Update

$$
\begin{aligned}
K_k &= P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \\
\delta\hat{x}_k^+ &= \hat{x}_{k-1}^+ - \hat{x}_k^- = K_k[y_k - h_k(\hat{x}_k^-, 0, 0)] \\
\hat{x}_k^+ &= \hat{x}_k^- + \delta\hat{x}_k^+ \\
P_k^+ &= (I - K_k H_k) P_k^- (I - K_k H_k)^T + K_k \tilde{R}_k K_k^T \\
\delta\hat{x}_k^+ &= 0 \qquad \text{(Reset)} \\
P_k^+ &= G P_k^+ G^T \qquad \text{(Update Covariance)}
\end{aligned}
$$

ESKF Algorithm: Reset

Reset Function G

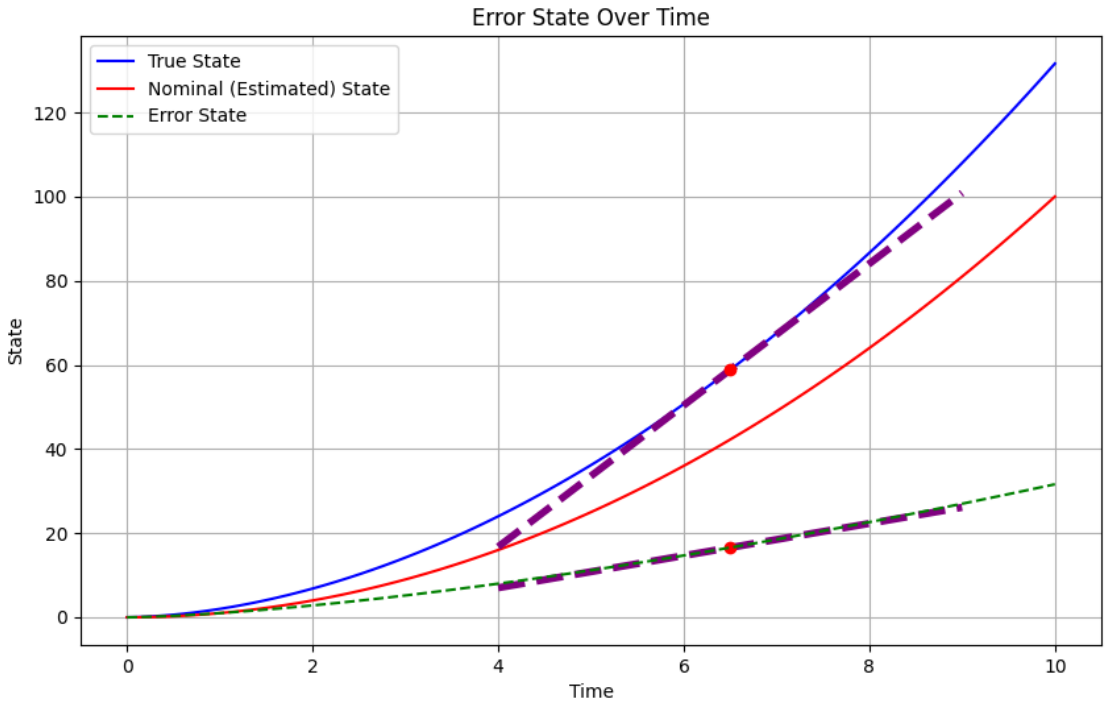
Once the nominal state is updated, the next step is to reset the value of the error state to 0. The reset is necessary because it is necessary to represent a new error for the new nominal state. The reset updates the covariance of the error state P_k^+.

$$g(\delta x) = \delta x - \delta \hat{x}_k^- \qquad \& \qquad G = \left. \frac{\partial g}{\partial \delta x} \right|_{\delta \hat{x}_k^+}$$

ESKF vs. EKF: Key Advantages

1. Superior Linearization

- **ESKF:** Always linearizes around the 'zero error' point ($\delta x = \mathbf{0}$). This is the most accurate possible point for a linear approximation.
- **EKF:** Linearizes around the current state estimate ($\hat{x}$), which may be inaccurate and lead to significant linearization errors.



ESKF vs. EKF: Key Advantages

2. Graceful Handling of Rotations

- **ESKF:** The full, non-linear rotation (quaternion) lives in the nominal state. The filter only deals with small, 3D error-angle vectors, which are mathematically simple and avoid singularities.
- **EKF:** Must linearize the quaternion manifold directly, which is more complex and can be less stable.

3. Computational Efficiency

- The computationally intensive non-linear propagation is separated from the filter.
- The Kalman filter itself operates on a small state vector, which is always well-behaved.

Section 5.2: Multi-State Constraint Kalman Filter (MSCKF)

Vision-Aided Inertial Navigation

"The primary contribution of this work is the derivation of a measurement model that is able to express the **geometric constraints** that arise when a static feature is observed from **multiple camera poses**."

Time Complexity

Computational complexity only linear in the number of features. $O(N)$

Original Paper

Mourikis, Anastasios I., and Stergios I. Roumeliotis. *"A multi-state constraint Kalman filter for vision-aided inertial navigation."* Proceedings 2007 IEEE international conference on robotics and automation (ICRA). IEEE, 2007.

IMU State and Error-State Definitions

IMU State Vector

$$X_{\text{IMU}} = \begin{bmatrix} {}^{G}_{I}\bar{q}^T & b_g^T & {}^{G}v_I^T & b_a^T & {}^{G}p_I^T \end{bmatrix}^T \tag{1}$$

where:

- ${}^{G}_{I}\bar{q}$ is the unit quaternion from the IMU $\{I\}$ to the global $\{G\}$.
- ${}^{G}v_I$ and ${}^{G}p_I$ are the IMU velocity and position in $\{G\}$.
- b_g and b_a are the gyroscope and accelerometer biases.

IMU Error-State Vector

The error state is standard additive error ($\tilde{x} = x - \hat{x}$) for position, velocity, and biases. For the quaternion, a multiplicative error is used ($\bar{q} = \delta\bar{q} \otimes \hat{\bar{q}}$).

$$\tilde{X}_{\text{IMU}} = \begin{bmatrix} \delta\theta^T & \tilde{b}_g^T & {}^{G}\tilde{v}_I^T & \tilde{b}_a^T & {}^{G}\tilde{p}_I^T \end{bmatrix}^T$$

The error quaternion $\delta\bar{q}$ is parameterized by the 3-element rotation vector $\delta\theta$.

Full State Vector (IMU + N Cameras)

Full State Vector

The full state vector **augments** the IMU state with the attitude and position of N cameras.

$$\hat{X}_k = \begin{bmatrix} \hat{X}_{\text{IMU}_k}^T & {}^{C_1}_{G}\hat{q}^T & {}^{G}\hat{p}_{C_1}^T & \dots & {}^{C_N}_{G}\hat{q}^T & {}^{G}\hat{p}_{C_N}^T \end{bmatrix}^T$$

where ${}^{C_i}_{G}\hat{q}$ and ${}^{G}\hat{p}_{C_i}$ are the estimated attitude and position of camera i with respect to the global frame $\{G\}$.

Full Error-State Vector

The EKF error-state vector is defined accordingly:

$$\tilde{X}_k = \begin{bmatrix} \tilde{X}_{\text{IMU}_k}^T & \delta\theta_{C_1}^T & {}^{G}\tilde{p}_{C_1}^T & \dots & \delta\theta_{C_N}^T & {}^{G}\tilde{p}_{C_N}^T \end{bmatrix}^T$$

This vector concatenates the IMU error-state with the orientation and position errors for each of the N cameras.

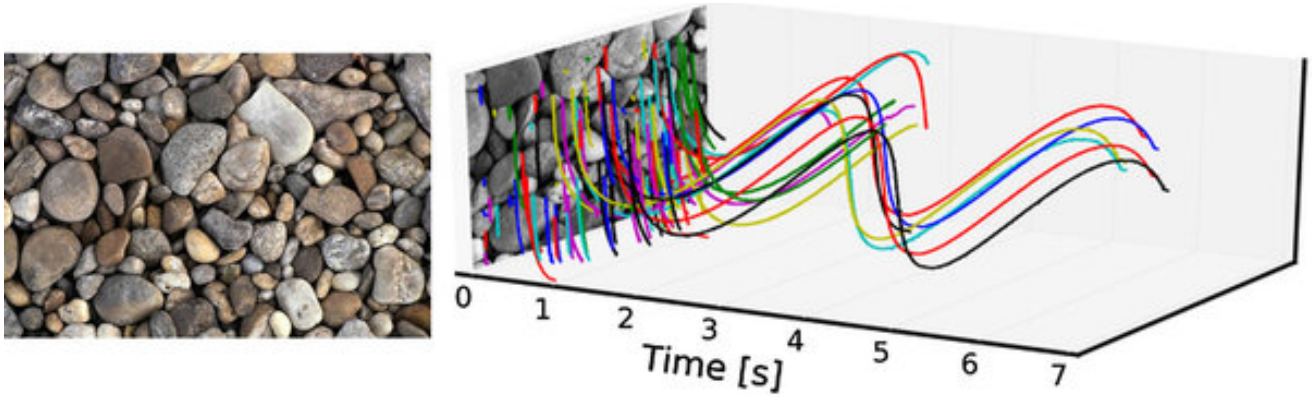
Predicion&Update Cycle

Predict: Standard Error-State INS Mechanism

The filter propagation equations are derived by discretization of the continuous-time IMU system model.

Update: Main Contribution

The camera observations are **grouped per tracked feature**, rather than per camera pose where the measurements were recorded. All the measurements of the same 3D point are used to define a **constraint equation**, relating all the camera poses at which the measurements occurred.This is achieved *without* including the feature position in the filter state vector.



Constraint

The constraint imposed by measurement matrix H_k.

1. Define residual $\mathbf{r}^{(j)} = \mathbf{y}^{(j)} - \hat{\mathbf{y}}^{(j)}$ for feature f_j
2. Linearize it: $\mathbf{r}^{(j)} \approx \mathbf{H}_X^{(j)} \tilde{\mathbf{X}} + \mathbf{H}_f^{(j)} {}^G\tilde{\mathbf{p}}_{f_j} + \mathbf{v}^{(j)}$

The residual $\mathbf{r}^{(j)}$ is not in the classical measurement form, and cannot be directly applied for measurement updates in the EKF. To overcome this problem, define a residual $\mathbf{r}_o^{(j)}$, by projecting $\mathbf{r}^{(j)}$ on the left nullspace of the matrix $\mathbf{H}_f^{(j)}$. Specifically, if we let $\mathbf{A}$ denote the unitary matrix whose columns form the basis of the left nullspace of $\mathbf{H}_f$, we obtain:

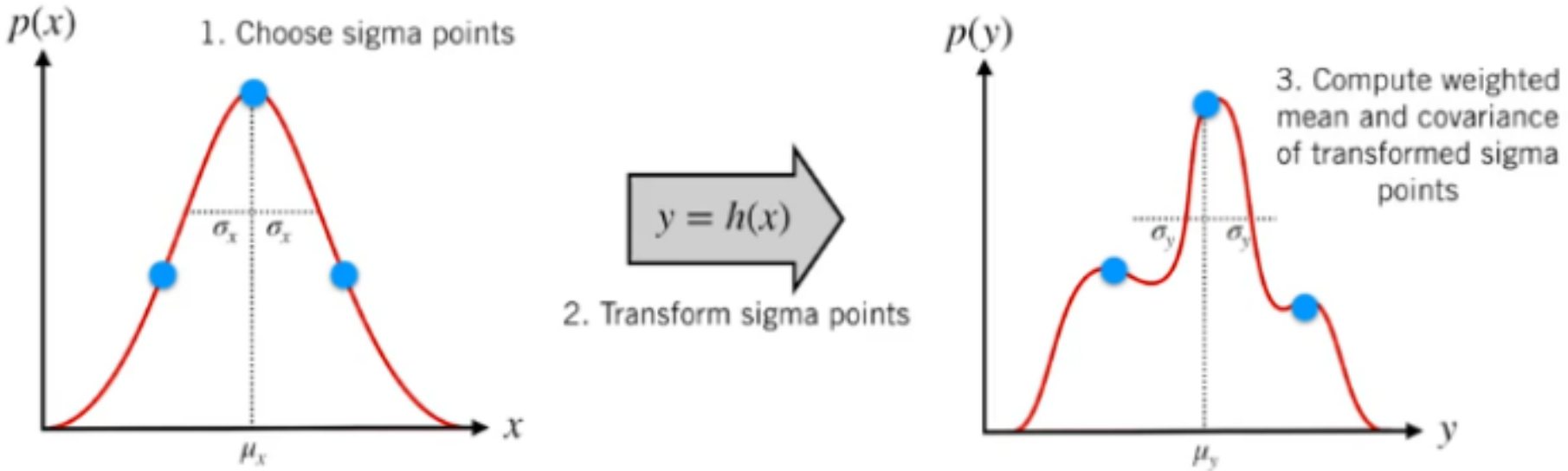
$$\mathbf{r}_o^{(j)} = \mathbf{A}^T(\mathbf{z}^{(j)} - \hat{\mathbf{z}}^{(j)}) \approx \mathbf{A}^T \mathbf{H}_X^{(j)} \tilde{\mathbf{X}} + \mathbf{A}^T \mathbf{n}^{(j)} \tag{23}$$

$$= \mathbf{H}_o^{(j)} \tilde{\mathbf{X}}^{(j)} + \mathbf{n}_o^{(j)} \tag{24}$$

Section 5.3: Unscented Kalman Filter (UKF)

Core Principle

It's easier to approximate a **probability distribution** than it is to approximate a **nonlinear function**.



Unscented Transform

The UKF(a.k.a. Sigma Point Kalman Filter) uses the **Unscented Transform**:

1. **Select Sigma Points:** A small, carefully chosen set of points (**sigma points**) are generated so that their mean and covariance exactly match the current state estimate.
2. **Propagate the Points:** Each sigma point is passed directly through the **true nonlinear functions** (f and h). No linearization or Jacobians are needed.
3. **Recover the New Distribution:** The mean and covariance of the resulting transformed points are calculated. This new distribution becomes the updated state estimate, which more accurately captures the true outcome.

UKF: Goal & Advantages over EKF

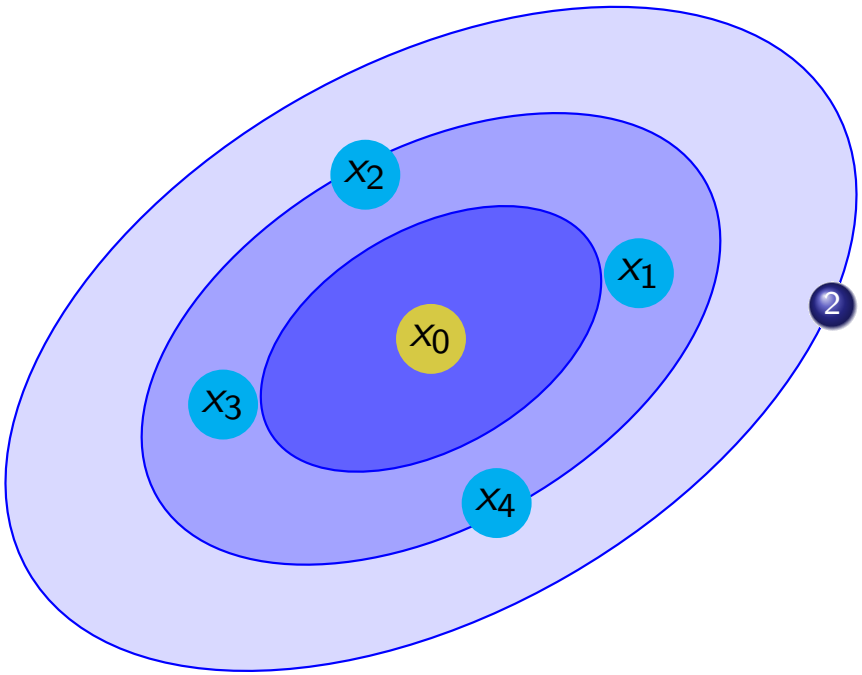
It holds several key advantages over the Extended Kalman Filter (EKF):

- **Higher Order Accuracy (Mean):** The estimated mean and covariance is correct up to the **third order** of the Taylor series expansion of the nonlinear function (due to nature of Unscented Transform).
  - The EKF's linearization approach is only accurate to the **first order**.
  - This results in a much smaller error compared to the EKF's linear approximation ($\mathbf{HPH}^T$), which omits these terms.
- **No Jacobian Matrices:** The UKF has a significant computational advantage in that it **does not require the derivation or calculation of Jacobian matrices** ($\mathbf{H}$).
  - This simplifies implementation, especially for highly complex nonlinear models.

UKF | Prediction step

Choosing Sigma Points

For an N-dimensional PDF $\mathcal{N}(\mu_x, \Sigma_{xx})$, we need $2N+1$ sigma points:



$\mathcal{N}(\mu_x, \Sigma_{xx})$

1. Compute the Cholesky Decomposition of the covariance matrix

$$\mathbf{L}\mathbf{L}^T = \Sigma_{xx} \quad (\mathbf{L} \text{ lower triangular})$$

2. Calculate the sigma points

$$\mathbf{x}_0 = \mu_x$$
$$\mathbf{x}_i = \mu_x + \sqrt{N+\kappa}\,\mathrm{col}_i\mathbf{L} \quad i = 1, \ldots, N$$
$$\mathbf{x}_{i+N} = \mu_x - \sqrt{N+\kappa}\,\mathrm{col}_i\mathbf{L} \quad i = 1, \ldots, N$$

UKF | Prediction step

Forward Pass and New Mean & Covariance

3-) Next we pass each of our $2N+1$ sigma points through the nonlinear function $h(\mathbf{x})$

$$\mathbf{y}_i = h(\mathbf{x}_i) \qquad i = 0, \ldots, 2N$$

4-) And finally compute the mean and covariance of the output PDF

$$\text{Mean:} \quad \boldsymbol{\mu}_y = \sum_{i=0}^{2N} \alpha_i \mathbf{y}_i$$

$$\text{Covariance:} \quad \boldsymbol{\Sigma}_{yy} = \sum_{i=0}^{2N} \alpha_i (\mathbf{y}_i - \boldsymbol{\mu}_y)(\mathbf{y}_i - \boldsymbol{\mu}_y)^T$$

$$\text{Weights:} \quad \alpha_i = \begin{cases} \dfrac{\kappa}{N+\kappa} & i = 0 \\ \dfrac{1}{2(N+\kappa)} & \text{otherwise} \end{cases}$$

UKF | Correction step

To correct the state estimate using measurements at time k, use the nonlinear measurement model and the sigma points from the prediction step to predict the measurements.

1. Predict measurements from propagated sigma points

$$\mathcal{Y}_k^{(i)} = \mathbf{h}(\mathcal{X}_k^{(i)}, \mathbf{0}) \quad i = 0...2N$$

2. Estimate mean and covariance of predicted measurements

$$\hat{\mathbf{y}}_k = \sum_{i=0}^{2N} \alpha^{(i)} \mathcal{Y}_k^{(i)}$$

$$\mathbf{P}_{y_k} = \sum_{i=0}^{2N} \alpha^{(i)} (\mathcal{Y}_k^{(i)} - \hat{\mathbf{y}}_k)(\mathcal{Y}_k^{(i)} - \hat{\mathbf{y}}_k)^T + \mathbf{R}_k$$

UKF | Correction step

3. Compute cross-covariance and Kalman gain

$$\mathbf{P}_{x_k y_k} = \sum_{i=0}^{2N} \alpha^{(i)} (\mathcal{X}_k^{(i)} - \hat{\mathbf{x}}_k^-)(\mathcal{Y}_k^{(i)} - \hat{\mathbf{y}}_k)^T$$

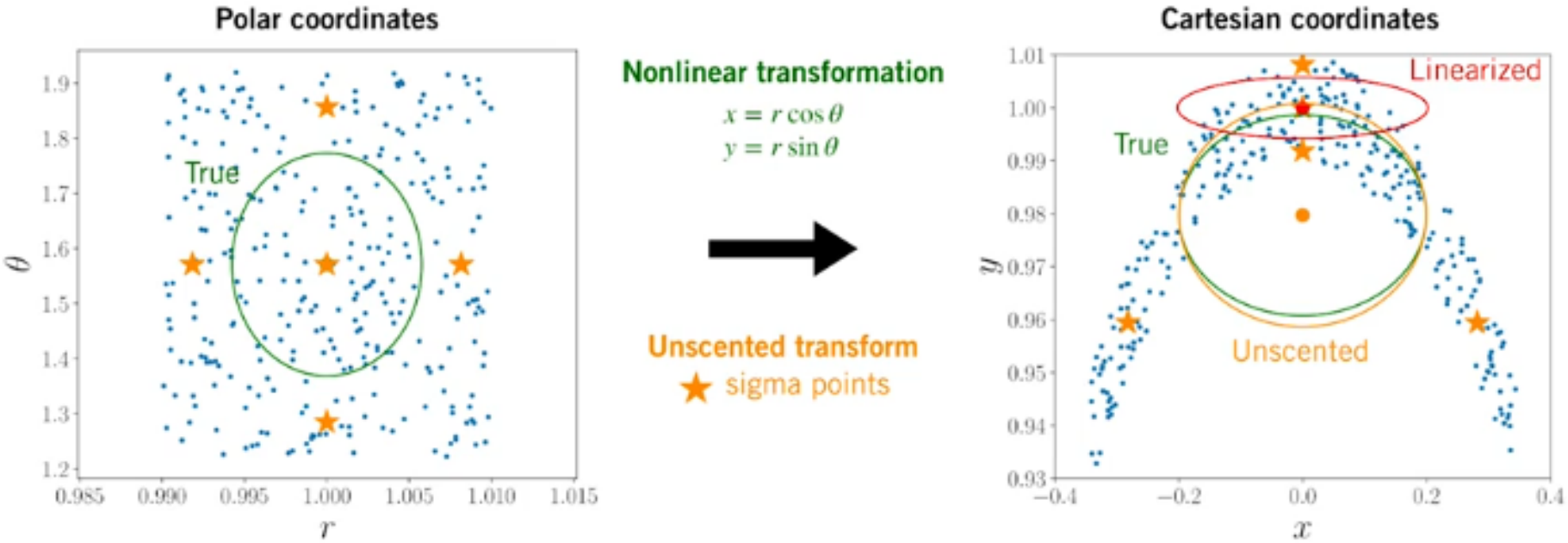
$$\mathbf{K}_k = \mathbf{P}_{x_k y_k} \mathbf{P}_{y_k}^{-1}$$

4. Compute corrected mean and covariance

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{y}_k - \hat{\mathbf{y}}_k)$$

$$\hat{\mathbf{P}}_k^+ = \hat{\mathbf{P}}_k^- - \mathbf{K}_k \mathbf{P}_{y_k} \mathbf{K}_k^T$$

UKF vs Linearization



UKF captures the nonlinear data better than linearized KF.

Section 5.4: Particle Filter

A completely nonlinear state estimator.
The particle filter is a completely nonlinear, statistical, brute-force approach to estimation that often works well for problems that are difficult for the conventional Kalman filter (i.e., systems that are highly nonlinear).

Has many other names:

Including sequential importance sampling, bootstrap filtering, the condensation algorithm, interacting particle approximations, Monte Carlo filtering, sequential Monte Carlo (SMC) filtering, and survival of the fittest.

History and Computational Power

Particle filters suggested in the 1940s but only since the 1980s has computational power been adequate for their implementation. Even now it is the computational burden of the particle filter that is its primary obstacle to more widespread use.

Implementation of Bayesian State Estimation

The particle filter was invented to numerically implement the Bayesian estimator.

1. Assuming that the pdf of the initial state $p(x_0)$ is known, randomly generate N initial particles on the basis of the pdf$p(x_0)$. These particles are denoted $x_{0,i}^{+}(i = 1, \cdots, N)$.
2. For k = 1,2, $\cdots$:
   3. $x_{k,i}^{-} = f_{k-1}(x_{k-1,i}^{+}, w_{k-1}^{i}) \qquad (i = 1, \ldots, N)$
   4. Compute likelihood q_i of particle $x_{k,i}^{-}$
   5. Scale likelihood $q_i = \frac{q_i}{\sum_{j=1}^{N} q_j}$
   6. Generate set of *a posteriori* particles $x_{k,i}^{i}$ on the basis of relative likelihoods q_i.
   7. Set of particles $x_{k,i}^{i}$ is distributed accordingly pdf$p(x_k|y_k)$. Compute any desired statistical measure.

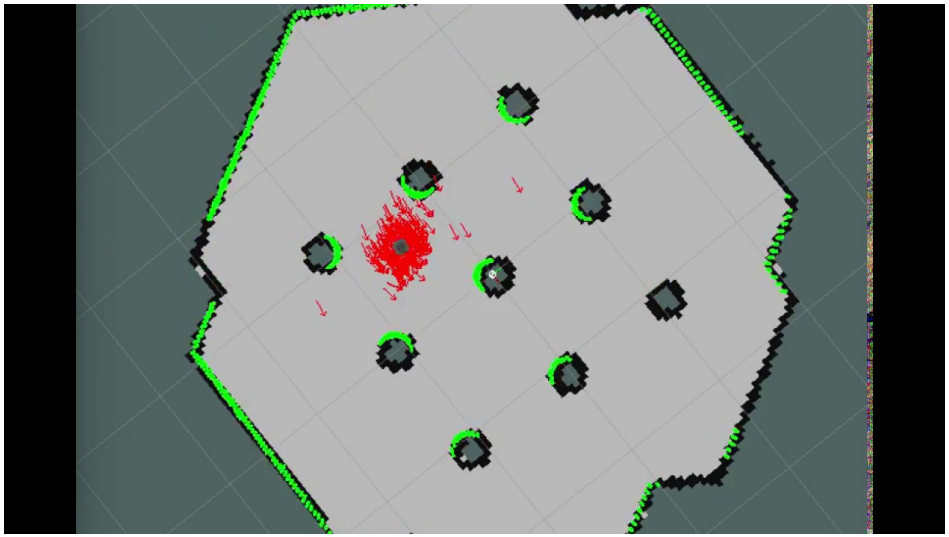
Popular Applications in ROS

Two popular elements of ROS Navigation Stack utilize Particle Filters:

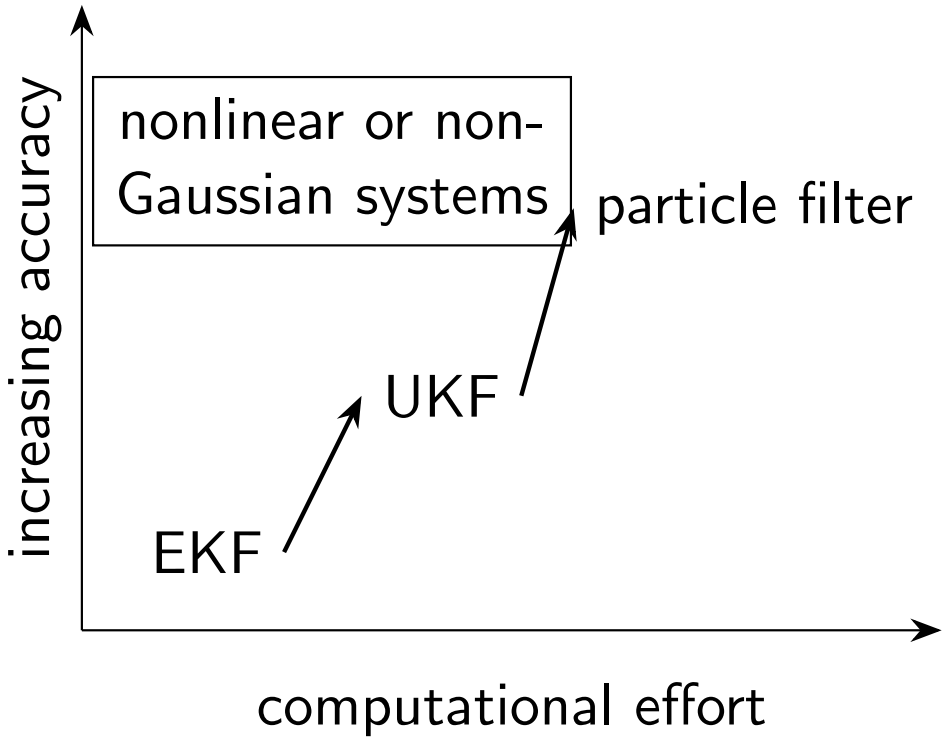
- Adaptive Monte Carlo Localization (AMCL)
- GMapping SLAM

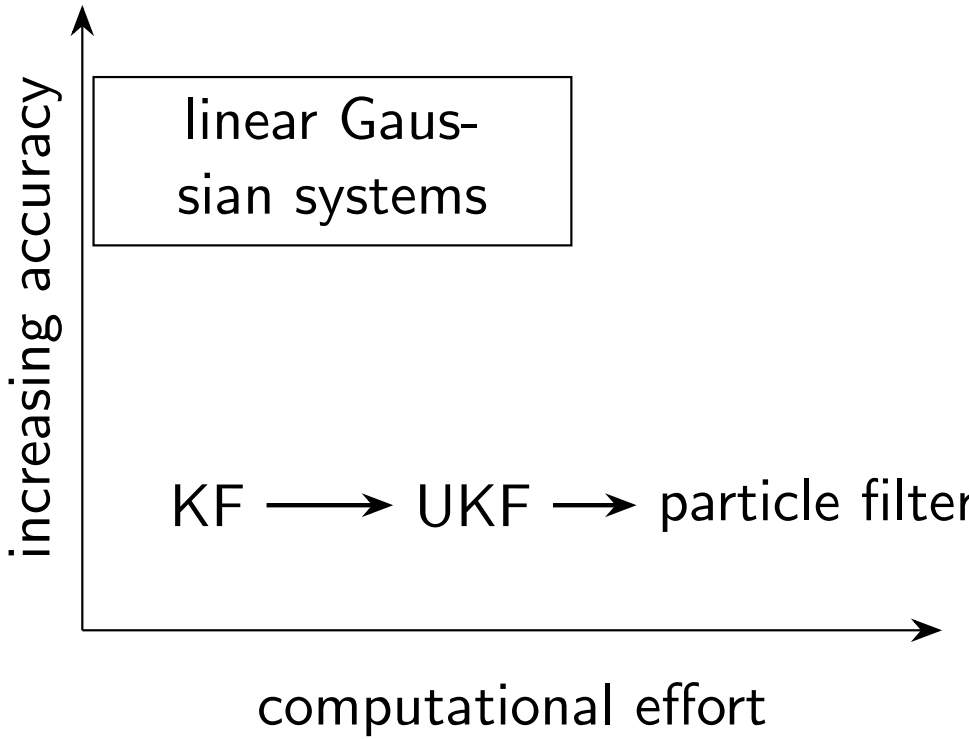
Practical Usage

1. Estimate the initial position of the robot with AMCL given the user's best estimate.
2. Localization during operation conducted via Robot Localization, which is an EKF/UKF based method.



Computational Effort Comparison





What We Have Covered Today?

1. Optimization basics: Least Square Method
2. Recursive Least Square as a noisy measurement update model
3. Dynamic State Propagation
4. Linear Kalman Filter
5. Extended (Nonlinear) Kalman Filter
6. Variations of KF (ESKF, MSCKF, UKF)
7. Introduction to Particle Filters

Summary

- State estimation is **the set of problems** of reconstructing the underlying state of a system given a sequence of measurements as well as a prior model of the system.
- The Kalman filter is a **powerful tool** for combining information and estimating state in the presence of uncertainty. It is optimal for linear systems with Gaussian noise.
- A successful KF requires system **dynamics** (like eqns. of motion or INS) and **measurement** (like VO) models as accurate as possible.
- All system dynamics include some degree of **noise**, thus, **uncertainty** due to high complexity of real-world.
- All sensors have a **limited precision**. Therefore, all measurements derived from real sensors have associated uncertainty. It is important to **keep track of all the uncertainties** involved and therefore (it is hoped) know how confident we can be in our estimate.
- There are infinitely-many **variations** of KF. Use whichever is suitable for your application.

Final Reminders

1-) Kalman Filter is a tool.

It is powerful, but still a **tool**. It can not be the goal itself. It helps you to achieve your goal.

2-) Know your dynamics well.

KF is meaningful only if your dynamics are well represented. Focus on deriving a robust **state transition** (f) and **measurement** (h) functions first. Then apply KF to make them work together.

3-) Check nonlinearity.

If your data is highly nonlinear, you will suffer from **linearization error**.

4-) Probabilities

Our noisy world makes everything a **random variable**. We do not find where the robot is, instead, we find what is the probability of robot being in a given position/orientation.

Pills Hard to Swallow

Does Kalman Filters Work ?

Well, it works for **2D planar** linear-like motion of robots.

What if non-planar highly non-linear motion ?

Not really. **Linearization error** is a real problem. Besides, today's level of KF has theoretical limitations when rotations are severe.

Does this help novel research ?

Yes. Many novel approaches are still compared to EKF based methods. Moreover, this is the most gentle introduction to the **state estimation**. More advanced methods are harder to get into without this background.

Is there still a EKF based research ?

Yes. Check this **CVPR** 2025 Highlight paper: EBS-EKF: Accurate and High Frequency Event-based Star Tracking

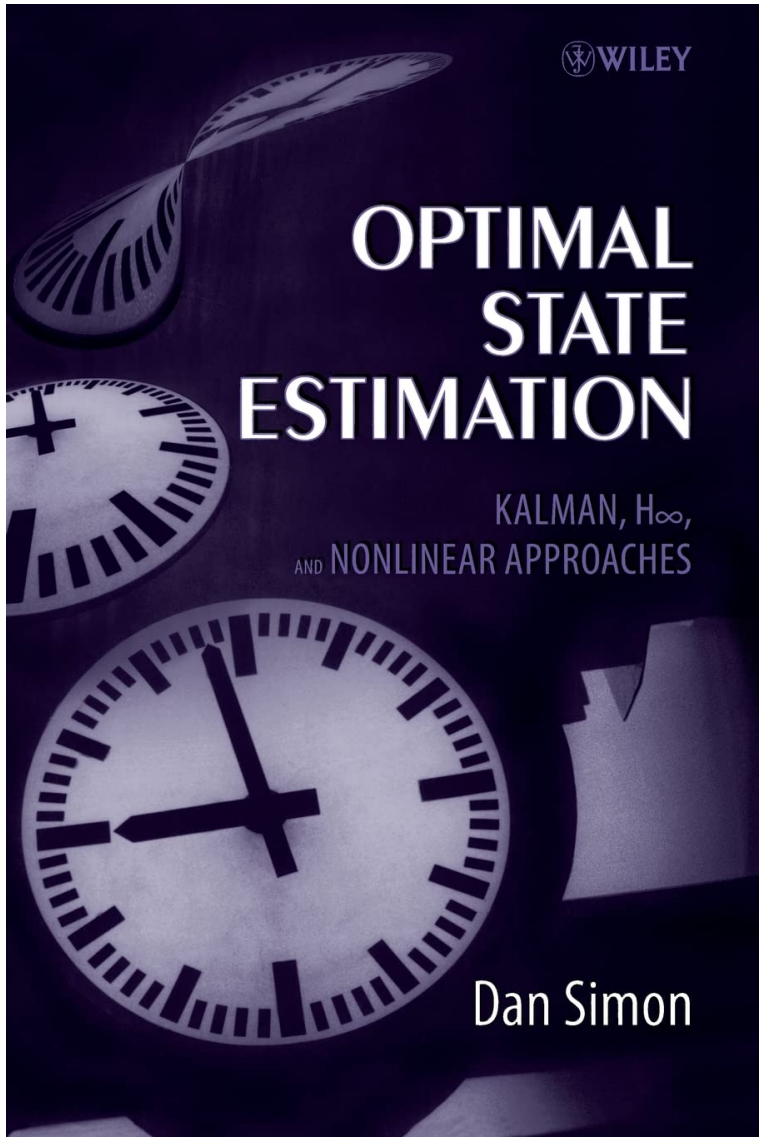
Advanced Topics in State Estimation

- **Probabilistic Robotics**: Same thigh as what we covered today but this approach more popular in the current research.
- **Lie Algebra**: State Estimation in 3D (Quaternions require more math!) Only affects the system and measurement models.
- **Factor-Graphs**: Modern SLAM and visual localization is based on graph theory.
- **SLAM**: I think it is the core problem in robotics. Camera/Lidar based SLAM requires solid computer vision background.

Credits: Today's Content

Today's content is mainly based on two sources:

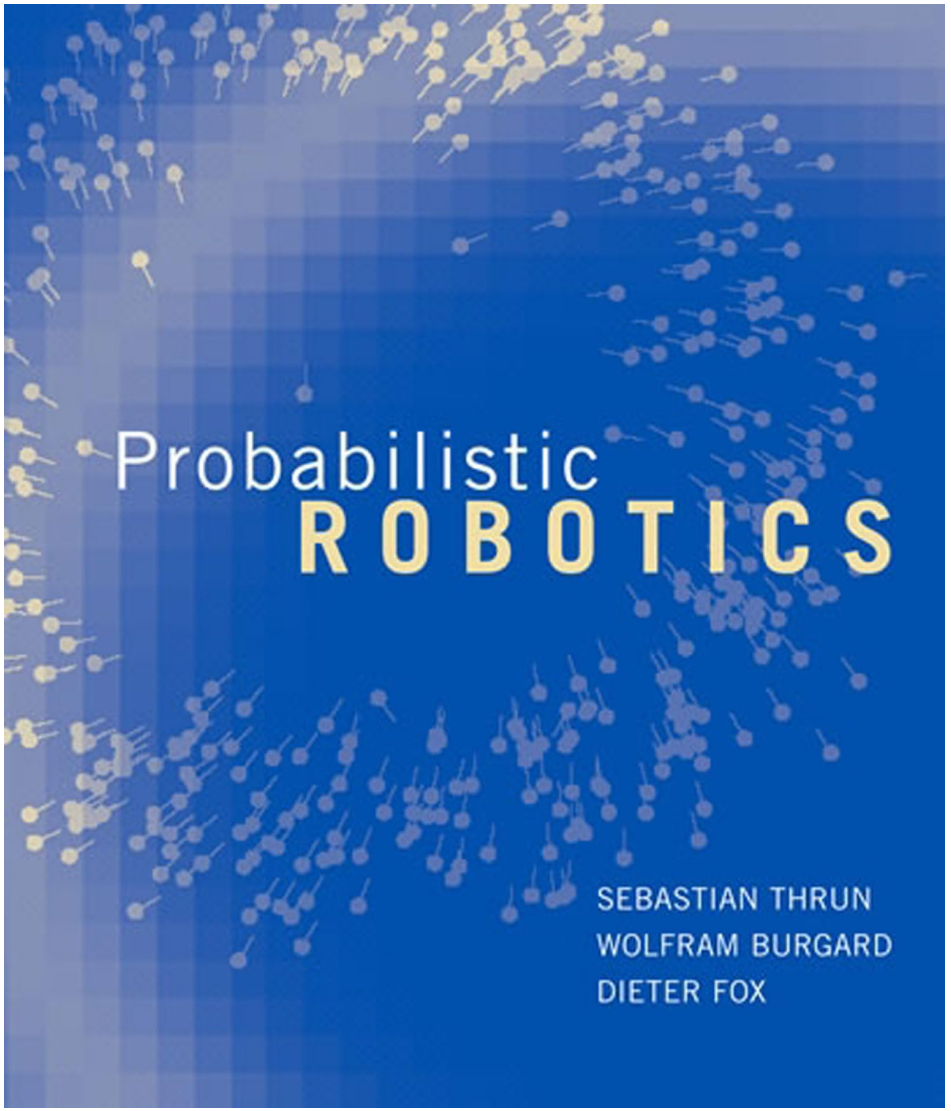
1. Simon, Dan. Optimal state estimation: Kalman, H infinity, and nonlinear approaches. John Wiley & Sons, 2006.
   - Least Square (Sections 3.1, 3.2, 3.3)
   - Dynamic Propagation (Chapter 4)
   - Kalman Filter (Sections 5.1, 5.2)
   - EKF (Sections 13.1, 13.2)
   - UKF (Sections 14.1, 14.2, 14.3)
   - Particle Filtering (Sections 15.1, 15.2)
   - Linear Systems and Probability Theories: Chapter 1 & 2
2. State Estimation and Localization for Self-Driving Cars, by University of Toronto, Coursera **(Module 3)**



Recommendation: Probabilistic Robotics

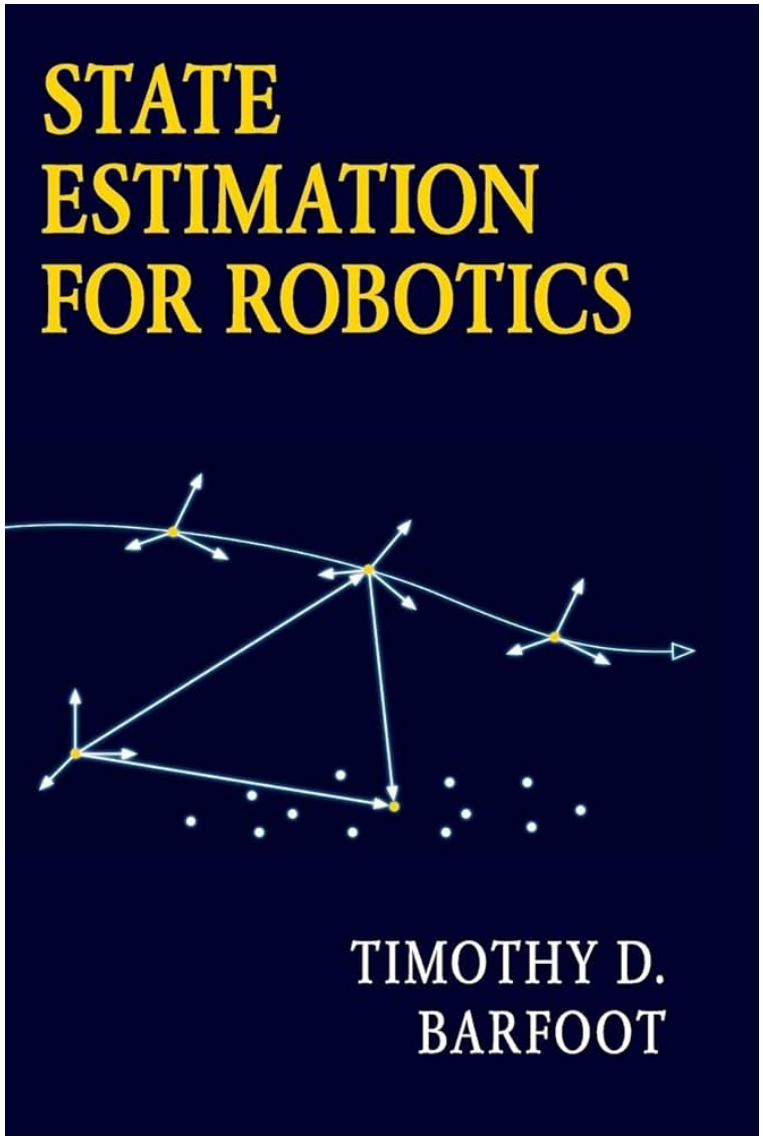
"Probabilistic Robotics" by Sebastian Thrun, Wolfram Burgard, and Dieter Fox

- Probability basics: Chapter 2



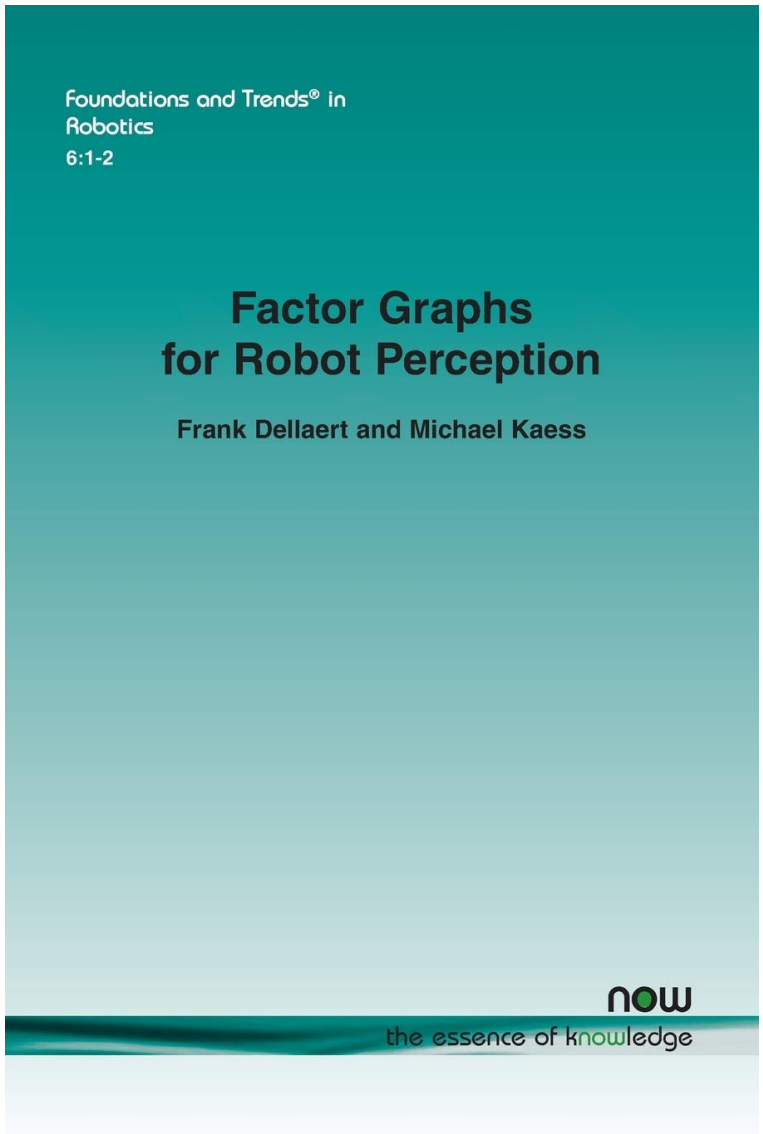
Recommendation: Lie Algebra

- "State Estimation for Robotics" by Timothy D. Barfoot (Parts II & III)
  - Probability theory: Chapter 1
  - KF and nonlinear KF: Chapters 2 & 3
- Additional Resource: Sola, Joan. "Quaternion kinematics for the error-state Kalman filter." arXiv preprint arXiv:1711.02508 (2017).



Recommendation: Factor Graphs

"Factor Graphs for Robot Perception" by Frank Dellaert and Michael Kaess



Recommendation: SLAM Handbook

SLAM Handbook

- A recent collaborative work by the scholars.
- Not a detailed theoretical foundation but good to remind or scan key concepts.

SLAM Handbook

From Localization and Mapping to Spatial Intelligence

Edited by

Luca Carlone, Ayoung Kim, Frank Dellaert,
Timothy Barfoot, and Daniel Cremers

IMPORTANT NOTE ON RELEASE:
© Cambridge University Press. No reproduction of any part may take place without the written permission of Cambridge University Press.

Authored by

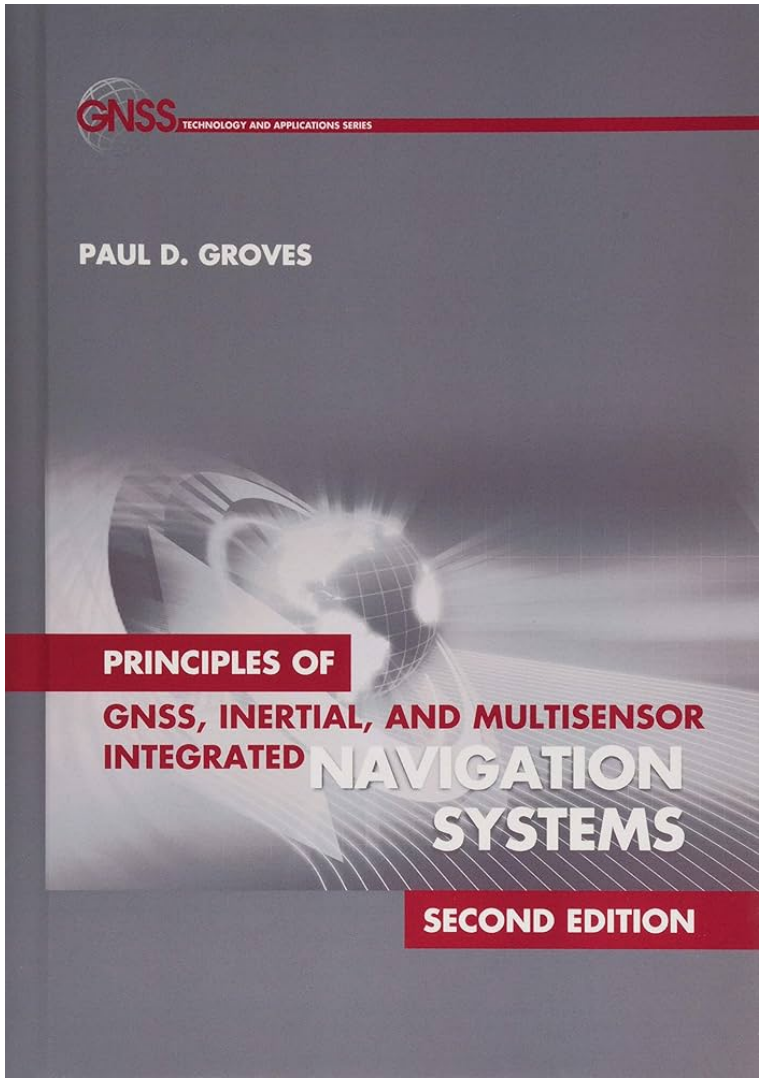
Henrik Andreasson
Timothy Barfoot
Martin Büchner
Luca Carlone
Margarita Chli
Daniel Cremers
Jia Deng
Jakob Engel
Cédric Le Gentil
Connor Holmes
Nathan Hughes
Kasra Khosoussi
John Leonard
Martin Magnusson
Matias Mattamala
Mustafa Mukadam
Helen Oleynikova
Marc Pollefeys
David Rosen
Jingnan Shi
Juan D. Tardós
Teresa Vidal-Calleja
Heng Yang
Shibo Zhao

Arash Asgharivaskasi
Jens Behley
Cesar Cadena
Yun Chang
Henrik Christensen
Andrew J. Davison
Gamini Dissanayake
Maurice Fallon
Christoffer Heckman
Guoquan Huang
Krishna Murthy Jatavallabhula
Ayoung Kim
Stefan Leutenegger
Joshua Mangelson
José M Martínez Montiel
Jose Neira
Lionel Ott
Victor Reijgwart
Davide Scaramuzza
Cyrill Stachniss
Zachary Teed
Chen Wang
Fu Zhang

Nikolay Atanasov
Jose Luis Blanco-Claraco
Marco Camurri
Boris Chidlovskii
Javier Civera
Frank Dellaert
Kevin Doherty
Guillermo Gallego
Javier Hidalgo-Carrió
Shoudong Huang
Michael Kaess
Giseop Kim
Dominic Maggio
Hidenobu Matsuki
Sacha Morin
Paul Newman
Liam Paull
Jerome Revaud
Lukas Schmid
Niko Sunderhauf
Abhinav Valada
Felix Wimbauer
Ji Zhang

Recommendation: IMU Navigation (INS)

"Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems" by Paul Groves (Chapters 5 & 6 & 14)



References

- **Kalman Filter**: Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems.
- **Reference Book**: Simon, Dan. Optimal state estimation: Kalman, H infinity, and nonlinear approaches. John Wiley & Sons, 2006.
- **Online Course**: State Estimation and Localization for Self-Driving Cars, by University of Toronto, Coursera
- **EKF-SLAM**: Dissanayake, MWM Gamini, et al. "A solution to the simultaneous localization and map building (SLAM) problem." IEEE Transactions on robotics and automation 17.3 (2001): 229-241.
- **ESKF:** Im, Gyubeom. "Notes on Kalman Filter (KF, EKF, ESKF, IEKF, IESKF)." arXiv preprint arXiv:2406.06427 (2024).
- **MSCKF:** Mourikis, Anastasios I., and Stergios I. Roumeliotis. *"A multi-state constraint Kalman filter for vision-aided inertial navigation."* Proceedings 2007 IEEE international conference on robotics and automation (ICRA). IEEE, 2007.

Other References

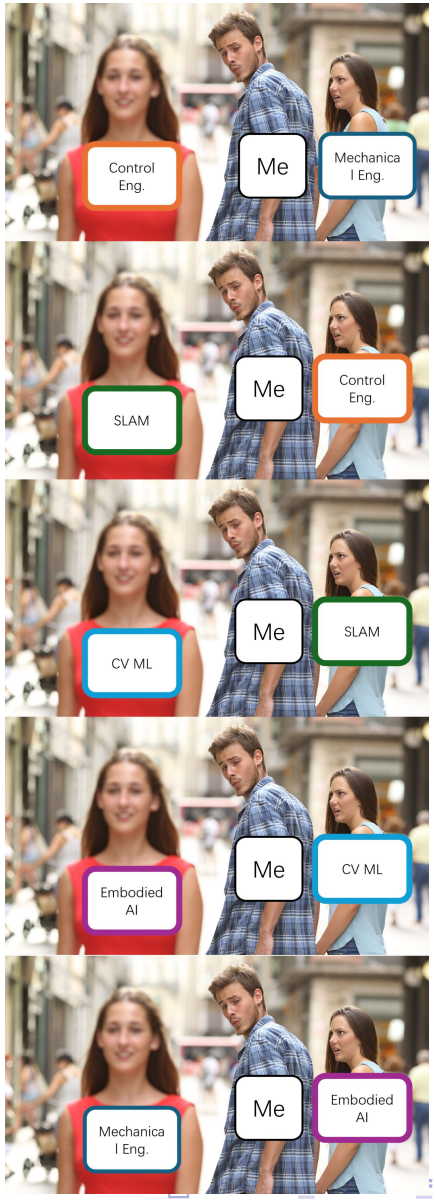
- **KF Blog:** A soft, yet effective blog for Kalman Filter: https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/
- **ESKF Paper**: Roumeliotis, Stergios I., Gaurav S. Sukhatme, and George A. Bekey. "Circumventing dynamic modeling: Evaluation of the error-state kalman filter applied to mobile robot localization." Proceedings 1999 IEEE International Conference on Robotics and Automation (ICRA). Vol. 2. IEEE, 1999.
- **AMCL**: Fox, Dieter. "KLD-sampling: Adaptive particle filters." Advances in neural information processing systems 14 (2001).
- **GMapping**: Grisetti, Giorgio, Cyrill Stachniss, and Wolfram Burgard. "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling." Proceedings of the 2005 IEEE international conference on robotics and automation (ICRA). IEEE, 2005.

What do you need to know in bigger picture?

Nearly everything.

Undying cycle of roboticist force you to know everything to some degree:

- Mechanical Engineering to design the physical body
- Control Engineering to make it move precisely
- SLAM to make robot navigate its environment
- Computer Vision and Machine Learning to make sense of what it "sees"
- Embodied AI to integrate intelligence directly with the physical form



Thank You!

Emre Girgin, 2025