Localization for Autonomous Driving

Brief tutorial to have fun with localization

Prepared by Oussama Code by Claude.ai

January 2025

I, Oussama EL HAMZAOUI confirm that the work presented in this report is my own (with the help of Claude.ai). Where information has been derived from other sources, I confirm that this has been indicated in the report.

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam et turpis gravida, lacinia ante sit amet, sollicitudin erat. Aliquam efficitur vehicula leo sed condimentum. Phasellus lobortis eros vitae rutrum egestas. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec at urna imperdiet, vulputate orci eu, sollicitudin leo. Donec nec dui sagittis, malesuada erat eget, vulputate tellus. Nam ullamcorper efficitur iaculis. Mauris eu vehicula nibh. In lectus turpis, tempor at felis a, egestas fermentum massa.

Acknowledgements

Interdum et malesuada fames ac ante ipsum primis in faucibus. Aliquam congue fermentum ante, semper porta nisl consectetur ut. Duis ornare sit amet dui ac faucibus. Phasellus ullamcorper leo vitae arcu ultricies cursus. Duis tristique lacus eget metus bibendum, at dapibus ante malesuada. In dictum nulla nec porta varius. Fusce et elit eget sapien fringilla maximus in sit amet dui.

Mauris eget blandit nisi, faucibus imperdiet odio. Suspendisse blandit dolor sed tellus venenatis, venenatis fringilla turpis pretium. Donec pharetra arcu vitae euismod tincidunt. Morbi ut turpis volutpat, ultrices felis non, finibus justo. Proin convallis accumsan sem ac vulputate. Sed rhoncus ipsum eu urna placerat, sed rhoncus erat facilisis. Praesent vitae vestibulum dui. Proin interdum tellus ac velit varius, sed finibus turpis placerat.

Contents

List of Figures

List of Tables

Abbreviations

ADAS	**A**dvanced **D**river **A**ssistance **S**ystems
CKF	**C**ubature **K**alman **F**ilter
EKF	**E**xtended **K**alman **F**ilter
GNSS	**G**lobal **N**avigation **S**atellite **S**ystem
HD	**H**igh **D**efinition (as in HD maps)
IMU	**I**nertial **M**easurement **U**nit
KF	**K**alman **F**ilter
LIDAR	**L**ight **D**etection and **R**anging
MCL	**M**onte **C**arlo **L**ocalization
PF	**P**article **F**ilter
RADAR	**R**adio **D**etection and **R**anging
RBPF	**R**ao-**B**lackwellized **P**article **F**ilter
SLAM	**S**imultaneous **L**ocalization and **M**apping
UKF	**U**nscented **K**alman **F**ilter
V2X	**V**ehicle-to-Everything Communication

Overview

Course chapters

Section to be deleted after completion of the couse

- ☒ Introduction to Localization
  - ☒ Problem statement and motivation
  - ☒ Types of localization problems
  - ☒ State estimation challenges
  - ☒ Sensor types and characteristics
  - ☒ Sources of uncertainty in robotics
- ☒ Probability Theory Foundations
  - ☒ Random variables and probability distributions
  - ☒ Bayes' theorem
  - ☒ Conditional probability
  - ☒ Markov assumption
  - ☒ Joint and marginal probabilities
  - ☒ Gaussian distributions
- ☐ Bayesian Filtering Framework
  - ☒ Recursive state estimation
  - ☒ Prediction step (motion model)
  - ☒ Update step (measurement model)
  - ☒ Chapman-Kolmogorov equation
  - ☒ Bayes filter algorithm
  - ☐ Linear vs nonlinear systems
- ☐ Kalman Filtering
  - ☐ Linear Kalman Filter
    - ☐ System model and assumptions
    - ☐ Prediction equations
    - ☐ Update equations
    - ☐ Uncertainty propagation
  - ☐ Extended Kalman Filter (EKF)
    - ☐ Linearization process
    - ☐ Jacobian matrices
    - ☐ Algorithm implementation
  - ☐ Unscented Kalman Filter (UKF)
    - ☐ Sigma points
    - ☐ Unscented transform
    - ☐ Algorithm implementation
- ☐ Particle Filtering
  - ☐ Monte Carlo methods

- ☐ Importance sampling
- ☐ Particle representation
- ☐ Sequential Importance Sampling (SIS)
- ☐ Resampling techniques
- ☐ Sample degeneracy and impoverishment
- ☐ Adaptive particle filtering

☐ Advanced Topics
- ☐ Multi-hypothesis tracking
- ☐ SLAM basics
- ☐ Sensor fusion techniques
- ☐ Loop closure
- ☐ Global vs local localization

☐ MATLAB Implementation: Kalman Filter
- ☐ Linear KF implementation
  - ☐ State prediction
  - ☐ Measurement update
  - ☐ Covariance propagation
- ☐ EKF implementation
  - ☐ System modeling
  - ☐ Jacobian computation
  - ☐ Filter implementation
- ☐ Visualization and analysis
- ☐ Performance evaluation

☐ MATLAB Implementation: Particle Filter
- ☐ Basic PF framework
- ☐ Particle initialization
- ☐ Motion model implementation
- ☐ Measurement model
- ☐ Weight computation
- ☐ Resampling implementation
- ☐ Visualization tools
- ☐ Performance metrics

☐ Practical Applications
- ☐ Vehicle localization case studies
- ☐ Robot navigation examples
- ☐ Integration with mapping
- ☐ Real-world challenges
- ☐ Best practices and optimization

☐ Project Work
- ☐ Implementation exercises
- ☐ Real dataset analysis
- ☐ Performance comparison of different filters
- ☐ Parameter tuning
- ☐ Documentation and presentation

Objectives

Chapter 1

Introduction to Localization

- ☒ Introduction to Localization
  - ☒ Problem statement and motivation
  - ☒ Types of localization problems
  - ☒ State estimation challenges
  - ☒ Sensor types and characteristics
  - ☒ Sources of uncertainty in robotics

1.1 Problem Statement and Motivation

In robotics and autonomous systems, localization addresses a fundamental question: "Where am I?" This seemingly simple question underlies many complex challenges in autonomous navigation. Imagine waking up in an unfamiliar room – you would use visual cues, memory, and perhaps a map to determine your location. Robots face a similar challenge, but must solve it using sensors and algorithms rather than human intuition.

Localization serves as the cornerstone of autonomous navigation. Without accurate knowledge of its position, a robot cannot effectively plan paths, avoid obstacles, or complete assigned tasks. This becomes particularly critical in applications like autonomous vehicles, where position errors of even a few centimeters can have serious consequences.

1.2 Types of Localization Problems

We can categorize localization problems based on their initial conditions and objectives:

Position Tracking represents the simplest case, where we know the initial position and need to maintain an accurate estimate as the robot moves. Think of using GPS in your car – you start from a known location and track your movement.

Global Localization presents a more challenging scenario where the initial position is unknown. The robot must determine its position from scratch using available sensor information and a map. This is analogous to opening a ride-sharing app in an unfamiliar city and waiting for it to locate you.

The Kidnapped Robot Problem is the most challenging variant, where a well-localized robot is suddenly transported to an unknown location. While this may seem artificial, it tests a system's ability to recover from catastrophic failures or sensor malfunctions.

1.3 Sensor Types and Characteristics

Localization systems typically rely on multiple sensor types, each with distinct advantages and limitations:

- Proprioceptive Sensors measure internal state changes, such as wheel encoders that track rotation or inertial measurement units (IMUs) that detect acceleration and angular velocity. While these sensors provide high-frequency updates, they suffer from cumulative errors through a process called dead reckoning.
- Exteroceptive Sensors observe the external environment. These include:
  - LIDAR (Light Detection and Ranging) which creates detailed 3D scans of surroundings
  - Cameras that provide rich visual information but require sophisticated processing
  - RADAR which offers reliable distance measurements even in adverse weather
  - GNSS (Global Navigation Satellite System) which provides absolute position but may suffer from urban canyon effects and multipath errors

1.4 Sources of Uncertainty

Understanding uncertainty is crucial for robust localization. Several factors contribute to localization uncertainty:

Motion Uncertainty arises from imperfect robot control and environmental interactions. When a robot moves, wheel slippage, uneven terrain, and mechanical play all introduce errors between commanded and actual motion.

Measurement Uncertainty stems from sensor limitations and noise. For example, LIDAR measurements might be affected by reflective surfaces, while camera images can be distorted by varying lighting conditions.

Environmental Uncertainty relates to the dynamic nature of the real world. Moving objects, changing weather conditions, and modifications to the environment can all affect localization accuracy.

Model Uncertainty comes from our simplified representations of complex physical systems. Our mathematical models of robot motion and sensor behavior are approximations that introduce additional uncertainty.

1.5 The Role of Probability Theory

Given these uncertainties, deterministic approaches to localization often fail in real-world conditions. This necessitates a probabilistic framework that can: - Represent and propa-

gate uncertainty through mathematical models - Fuse information from multiple, imperfect sensors - Handle conflicting measurements and outliers - Provide confidence estimates along with position estimates

This probabilistic approach leads us naturally to the Bayesian filtering framework, which we'll explore in subsequent chapters. The framework provides a mathematical foundation for combining prior knowledge, motion predictions, and sensor measurements to maintain an estimate of the robot's position over time.

Understanding these foundational concepts is crucial as we progress to more advanced topics in localization. The challenges and considerations introduced here will inform our discussion of specific algorithms and implementations throughout the course.

Would you like me to elaborate on any of these sections or move on to the probability theory foundations?

Chapter 2

Probability Theory Foundations

2.1 Random variables and probability distributions

2.1.1 Random Variables

A random variable is a mathematical way to describe outcomes of a random process. Think of it as a function that assigns a numerical value to each possible outcome of an experiment or observation.

Let's consider a practical example from robotics: imagine a robot's sensor measuring the distance to a wall. Even when the robot and wall are stationary, repeated measurements might give slightly different values due to sensor noise. Each measurement is a realization of a random variable that we could call "measured distance."

Random variables come in two main types:

Discrete random variables can only take specific, countable values. For instance, if we count the number of landmarks a robot sees in its field of view, this would be a discrete random variable - we can only see 0, 1, 2, or some whole number of landmarks.

Continuous random variables can take any value within a continuous range. Most sensor measurements in robotics are continuous random variables. Our distance sensor example could theoretically return any real number within its measurement range.

2.1.2 Probability Distributions

A probability distribution describes how likely each possible value of a random variable is to occur. It tells us the complete story of the random variable's behavior.

For discrete random variables, we use a Probability Mass Function (PMF). The PMF gives the probability of each possible value directly. For example, if we're counting landmarks:

- $P(X = 0) = 0.1$ (10% chance of seeing no landmarks)
- $P(X = 1) = 0.3$ (30% chance of seeing exactly one landmark)
- $P(X = 2) = 0.4$ (40% chance of seeing exactly two landmarks)

And so on…

For continuous random variables, we use a Probability Density Function (PDF). The PDF works differently because with continuous variables, the probability of getting any exact value is actually zero! Instead, the PDF gives us the relative likelihood of values occurring, and we integrate it over ranges to get probabilities.

The most important continuous probability distribution in robotics is the Gaussian (or Normal) distribution. It's defined by two parameters:

- (mu): the mean, representing the central value
- (sigma): the standard deviation, representing the spread

The Gaussian distribution appears naturally in many robotics scenarios because of the Central Limit Theorem. When many small random effects add up - like multiple sources of sensor noise - their combined effect tends to follow a Gaussian distribution.

In the context of localization, probability distributions help us represent:

1. The robot's belief about its position (often as a Gaussian in simple cases)
2. Uncertainty in sensor measurements
3. Noise in motion commands and their execution
4. The likelihood of different measurements given a particular position

Understanding these distributions is crucial because localization algorithms like Kalman filters and particle filters essentially manipulate these probability distributions to maintain and update the robot's position estimate over time.

2.2 Bayes theorem

Let me explain Bayes' theorem in a way that will make intuitive sense, starting with a simple example and then building up to its use in robotics.

Imagine you're a robot in a room, and you have a simple distance sensor. Sometimes your sensor shows a reading of 2 meters, but you're not sure if you're actually 2 meters from a wall or if your sensor is giving you a wrong reading.

To understand Bayes' theorem, let's break this situation down into pieces:

First, let's define what we know: - You might be 2 meters from a wall (we'll call this your "position") - Your sensor gives you a measurement of 2 meters (we'll call this your "measurement")

Now, what Bayes' theorem helps us figure out is: Given that your sensor reads 2 meters, what's the probability that you're actually 2 meters from the wall?

Here's the magic formula (don't worry, we'll break it down):

```
P(position | measurement) = P(measurement | position) × P(position) / P(
    measurement)
```

Let's understand each piece:

1. `P(position | measurement)` is what we want to know: the probability of being at a position, given our sensor measurement. This is called the "posterior probability."

2. `P(measurement | position)` is how likely we are to get this measurement if we really are at that position. We know our sensor isn't perfect - maybe it's 90% accurate when we're actually at 2 meters. This is called the "likelihood."

3. `P(position)` is what we believed about our position before taking the measurement. Maybe based on our last estimate, we thought there was a 70% chance we were at 2 meters. This is called the "prior probability."

4. `P(measurement)` is how likely we are to get this measurement in general. Think of it as a normalizing factor that makes all our probabilities add up to 100%.

Let's put some numbers in: - If our sensor is 90% accurate: P(measurement | position) = 0.9 - If we thought we were probably at 2m: P(position) = 0.7 - Let's say P(measurement) = 0.8 (this is calculated considering all possibilities)

Then:

```
P(position | measurement) = 0.9 × 0.7 / 0.8 = 0.79
```

This tells us that after getting the measurement, we're 79% confident about our position - more confident than our prior belief of 70%!

The beautiful thing about Bayes' theorem is that it gives us a formal way to: 1. Start with what we believe (prior) 2. Consider new evidence (likelihood) 3. Update our belief (posterior)

In robotics, we use this process continuously. Every time we: - Move (this changes our prior belief) - Take a measurement (this gives us new evidence) - We use Bayes' theorem to update our belief about where we are

This is the foundation of probabilistic robotics and the basis for algorithms like Kalman filters and particle filters, which we'll explore later.

2.3 Conditional probability

Think of probability as measuring how likely something is to happen. Now, conditional probability takes this a step further by asking: "How likely is this event to happen, given that we already know something else has happened?"

The formal notation for conditional probability is $P(A|B)$, which reads as "the probability of A given B." Mathematically, it's expressed as:

$$P(A|B) = P(A \cap B)/P(B)$$

Let's break this down with a real-world example. Imagine we have a deck of 52 playing cards, and we want to know the probability of drawing a king, given that we've already drawn a red card.

To solve this:

1. First, we identify what we know: we've drawn a red card (this is our condition B)

2. We want to find the probability of having a king among these red cards (this is our event A)

3. $P(B)$ = probability of drawing a red card = $26/52 = 1/2$
4. $P(A \cap B)$ = probability of drawing a red king = $2/52 = 1/26$
5. Therefore, $P(A|B) = (2/52)/(26/52) = 2/26 = 1/13$

This shows us something interesting: while the probability of drawing a king from the full deck is 4/52 (about 0.077), the probability of drawing a king given that we know the card is red is 1/13 (about 0.077). In this case, knowing the card is red didn't change the probability of it being a king, because kings are evenly distributed between red and black cards.

This leads us to an important concept: independence. If knowing one event doesn't affect the probability of another event, we say these events are independent. In such cases, $P(A|B) = P(A)$. However, in many real-world scenarios, events are dependent, and conditional probability helps us account for this dependency.

Consider a medical example: the probability of having a certain disease might be 1% in the general population, but if we know a person has a specific symptom, the conditional probability of having the disease given this symptom might be much higher, say 30%. This is why doctors use symptoms to update their diagnostic probabilities.

2.4 Markov assumption

The Markov assumption, also known as the Markov property, is a fundamental concept in probability theory that helps us model complex sequences of events in a manageable way. Let me break this down step by step.

The core idea of the Markov assumption is that the future state of a system depends only on its present state, not on its past states. In probability terms, this means that if we want to predict what happens next, we only need to know what's happening right now, not the entire history of what happened before.

To understand this more concretely, imagine you're watching the weather. A pure Markov process would say that tomorrow's weather only depends on today's weather, not on what the weather was like last week or last month. While this might seem like an oversimplification (and in reality, weather patterns are more complex), this assumption often proves surprisingly useful in many real-world applications.

Let's express this mathematically. For a sequence of events $X_1, X_2, X_3, ..., X_n$ the Markov property states that:

$$P(X_{n+1}|X_n, X_{n-1}, ..., X_1) = P(X_{n+1}|X_n)$$

This equation tells us that the probability of the next state (X_{n+1}) given all previous states is equal to the probability of the next state given just the current state (X_n). This dramatically simplifies our calculations while still capturing many important patterns in real-world processes.

Think of it like playing a game of chess. While each position arose from a long sequence of moves, a player really only needs to look at the current board position to

decide their next move. The specific sequence of moves that led to this position, while interesting historically, isn't directly relevant to choosing the next best move.

The Markov assumption is particularly powerful because it allows us to build practical models of complex systems. It's used in:

1. Natural Language Processing - In simple language models, the probability of the next word might depend only on the current word (or last few words), not the entire sentence history.
2. Financial Markets - Some basic models assume that tomorrow's stock price depends only on today's price, not the entire price history.
3. Biology - Gene sequences can be modeled using Markov chains, where each base pair depends only on the previous few pairs.
4. Machine Learning - Hidden Markov Models use this property to model sequential data in a computationally efficient way.

It's important to note that the Markov assumption comes in different "orders." What I've described is a first-order Markov process, where we only look at the immediate previous state. In a second-order Markov process, we look at the last two states, and so on. Higher-order Markov processes can capture more complex dependencies but require more computational resources.

This assumption, while powerful, isn't always perfectly accurate in real-world situations. Many processes have longer-term dependencies that a simple Markov model might miss. However, the simplification it provides often outweighs these limitations, making it an invaluable tool in probability theory and its applications.

2.5 Joint and marginal probabilities

Let me explain joint and marginal probabilities through an intuitive progression, starting with the fundamentals and building up to how they work together.

Joint probability represents the likelihood of two (or more) events occurring together. We write this as $P(A, B)$ or $P(A \cap B)$, which reads as "the probability of A and B happening." Think of it as the overlap in a Venn diagram - the space where both events occur simultaneously.

Let's make this concrete with an example. Imagine we're looking at weather data for a year. Let's consider two events:

- Event A: It's a cold day (temperature below $50°F$)
- Event B: It's a rainy day

The joint probability $P(A, B)$ would tell us the probability that a randomly chosen day is both cold AND rainy. If 73 days out of 365 were both cold and rainy, the joint probability would be $73/365 = 0.2$, or 20%.

Now, marginal probability is what we get when we're interested in the probability of just one event, regardless of what happens with other events. It's called "marginal"

because historically, these probabilities were written in the margins of probability tables. If we look at our weather example:

- $P(A)$: The probability of a cold day, regardless of rain
- $P(B)$: The probability of a rainy day, regardless of temperature

Here's where these concepts connect: marginal probabilities can be calculated by summing up joint probabilities. Mathematically:

$$P(A) = P(A, B) + P(A, notB)$$

In our weather example, if:

- 73 days were cold and rainy: $P(A, B) = 0.2$
- 109 days were cold and not rainy: $P(A, notB) = 0.3$

Then the marginal probability of a cold day $P(A) = 0.2 + 0.3 = 0.5$, or 50% of days.

We can visualize this with a probability table:

Table 2.1: Joint and Marginal Probabilities of Cold and Rainy Days. The sum of all joint probabilities equals 1, representing all possible weather combinations.

	Rainy (B)	Not Rainy	Marginal
Cold (A)	0.2	0.3	0.5
Not Cold	0.1	0.4	0.5
Marginal	0.3	0.7	1.0

Each cell shows a joint probability, and the margins show marginal probabilities. Notice how the marginals sum up the joint probabilities in their respective rows or columns.

This relationship between joint and marginal probabilities becomes especially important when working with conditional probabilities. Remember our earlier discussion about conditional probability? We can express it using joint and marginal probabilities:

$$P(A|B) = P(A, B)/P(B)$$

This shows how these concepts are deeply interconnected: joint probabilities help us calculate marginal probabilities, which in turn help us work with conditional probabilities.

Understanding these relationships is crucial in many real-world applications, from medical diagnosis (where we might look at joint probabilities of symptoms and diseases) to market analysis (examining the relationship between customer demographics and purchasing behaviors).

2.6 Gaussian distributions

We will dive into Gaussian distributions, also known as normal distributions, by building up from their fundamental characteristics to their broader significance in probability theory.

The Gaussian distribution is perhaps nature's most remarkable probability distribution. Picture a perfectly symmetrical bell-shaped curve - this is the visual signature of a Gaussian distribution. But why is this shape so special and ubiquitous?

At its core, a Gaussian distribution is defined by two key parameters:

1. The mean (μ) - the center point of the distribution, representing the average value
2. The standard deviation (σ) - which determines how spread out the values are from the mean

The mathematical formula for a Gaussian distribution is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

While this formula might look intimidating, we can understand its behavior through some key properties:

First, consider symmetry. The distribution is perfectly symmetrical around its mean, meaning values equally far above and below the mean are equally likely. This reflects many natural phenomena - for instance, human height variations around the average height.

Second, the "$68 - 95 - 99.7$ rule" tells us something remarkable about how probability is distributed:

- About 68% of values fall within one standard deviation of the mean
- About 95% fall within two standard deviations
- About 99.7% fall within three standard deviations

This leads us to an important insight: extreme values become exponentially less likely as we move away from the mean. Think about human height - while it's common to meet someone 2 inches taller than average, it's extremely rare to meet someone 12 inches taller.

The Gaussian distribution emerges naturally in many situations due to the Central Limit Theorem, which tells us that when we add up many independent random variables, their sum tends to follow a Gaussian distribution, regardless of the underlying distributions of the individual variables. This explains why we see Gaussian distributions so often in nature - many natural phenomena are the result of multiple small, independent effects adding together.

Let's consider some practical examples:

- Measurement Error: When scientists make repeated measurements of the same quantity, the errors typically follow a Gaussian distribution
- Financial Returns: Daily stock market returns often approximately follow a Gaussian distribution

- Biological Variations: Things like height, weight, and blood pressure in a population often follow roughly Gaussian distributions

Understanding Gaussian distributions is crucial for:

1. Statistical Testing: Many statistical tests assume underlying Gaussian distributions
2. Quality Control: Manufacturing processes often use Gaussian assumptions to set acceptable tolerance limits
3. Risk Analysis: Financial models frequently use Gaussian distributions to model market behavior
4. Machine Learning: Many algorithms assume Gaussian noise in their models

There's an interesting connection to earlier concepts we discussed: conditional probabilities involving Gaussian distributions have special properties. If you have two variables that follow a joint Gaussian distribution, the conditional distribution of one given the other is also Gaussian - a property that makes these distributions particularly useful in prediction problems.

Chapter 3

Bayesian Filtering Framework

- ☐ Bayesian Filtering Framework
  - ☐ Recursive state estimation
  - ☐ Prediction step (motion model)
  - ☐ Update step (measurement model)
  - ☐ Chapman-Kolmogorov equation
  - ☐ Bayes filter algorithm
  - ☐ Linear vs nonlinear systems

3.1 Recursive state estimation

At its heart, recursive state estimation is about continuously updating our belief about a system's state as new measurements come in. Imagine you're trying to track a moving object - at any moment, you want to know its position and velocity, but your sensors are noisy and the object's movement isn't perfectly predictable.

The recursive nature comes from how we update our estimate: instead of processing all past measurements every time we get new data, we maintain a current estimate and update it using only the newest measurement. This makes the process computationally efficient and suitable for real-time applications.

The mathematical framework follows two main steps that repeat over time:

1. Prediction Step (Time Update): First, we predict how the state will evolve based on our system model:

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

This integral combines our previous estimate $p(x_{k-1}|z_{1:k-1})$ with our motion model $p(x_k|x_{k-1})$ to predict the new state.

2. Correction Step (Measurement Update): When we get a new measurement, we update our prediction using Bayes' rule:

$$p(x_k|z_{1:k}) = \eta p(z_k|x_k)p(x_k|z_{1:k-1})$$

Here, $p(z_k|x_k)$ is our measurement model, and is a normalizing constant.

Let's make this concrete with an example. Imagine you're tracking a drone: - State (x): position and velocity - Measurements (z): GPS readings - Motion model: physics equations for how the drone moves - Measurement model: GPS error characteristics

At each time step: 1. You predict where the drone should be based on physics and your last estimate 2. You get a GPS reading 3. You combine your prediction with the measurement to get an updated estimate 4. Repeat for the next time step

The beauty of this approach is that it naturally handles uncertainty. Both your predictions and measurements come with uncertainty (represented as probability distributions), and the Bayesian framework tells you exactly how to combine these uncertainties to get your best estimate.

A key insight is that this framework maintains a complete probability distribution over possible states, not just a single "best guess." This gives you important information about how certain you are about your estimates.

The most famous implementation of this framework is the Kalman Filter, which assumes all uncertainties are Gaussian. This simplifying assumption makes the mathematics tractable while still being useful for many real-world applications. However, the framework itself is more general and can handle non-Gaussian distributions through techniques like particle filters.

3.2 Prediction step (motion model)

The prediction step is fundamentally about using our understanding of how a system evolves over time to estimate its future state. Think of it as using physics to predict where a ball will be in the next moment, given its current position and velocity.

The motion model is mathematically expressed as $p(x(k)|x(k-1)$, which represents the probability of transitioning from state x_{k-1} to state x_k. This probability encapsulates both our deterministic understanding of system dynamics and our uncertainty about random disturbances.

Let's break this down with a concrete example of tracking a moving vehicle in 2D space. Our state vector might include:

- Position (x, y)
- Velocity (vx, vy)
- Acceleration (ax, ay)

In its simplest form, the motion model might use basic physics equations:

$$
\begin{aligned}
x(k) &= x(k-1) + vx(k-1)\Delta t + \frac{1}{2}ax(k-1)\Delta t^2 \\
y(k) &= y(k-1) + vy(k-1)\Delta t + \frac{1}{2}ay(k-1)\Delta t^2 \\
vx(k) &= vx(k-1) + ax(k-1)\Delta t \\
vy(k) &= vy(k-1) + ay(k-1)\Delta t
\end{aligned}
$$

However, in real-world scenarios, we need to account for uncertainties. These might come from: 1. Process noise (random disturbances to the system) 2. Model imperfections (our equations aren't perfect) 3. External forces we can't measure directly

This is where the probabilistic nature of the motion model becomes crucial. We typically model these uncertainties using probability distributions. In many cases, we assume Gaussian noise, leading to a linear motion model of the form:

$$x(k) = F(k)x(k-1) + B(k)u(k) + w(k)$$

Where: - $F(k)$ is the state transition matrix (encoding our physics equations) - $B(k)u(k)$ represents known control inputs - $w(k)$ is the process noise (typically assumed Gaussian)

For our vehicle example, the state transition matrix $F(k)$ might look like:

$$\begin{bmatrix} 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

This matrix encodes how each state variable influences the others over time. Reading across each row tells us how we compute each new state variable.

The complete prediction step involves propagating not just the state estimate but also its uncertainty. If we're using a Gaussian representation, this means we need to update both the mean and covariance of our state estimate:

Mean prediction:

$$\hat{x}(k|k-1) = F(k)\hat{x}(k-1|k-1) + B(k)u(k)$$

Covariance prediction:

$$P(k|k-1) = F(k)P(k-1|k-1)F(k)^\top + Q(k)$$

Where Q(k) is the process noise covariance matrix that quantifies our uncertainty about the motion model.

This probabilistic approach allows us to: 1. Make predictions about future states 2. Maintain an estimate of our uncertainty 3. Account for both systematic and random effects 4. Prepare for the measurement update step where we'll combine these predictions with actual measurements

3.3 Update step (measurement model)

The measurement update (or correction step) is where we refine our predicted state estimate by incorporating new sensor measurements. This step is fundamentally based on Bayes' rule, which tells us how to update probabilities when we get new evidence.

The key equation for the measurement update is:

$$p(x_k|z_{1:k}) = \eta p(z_k|x_k)p(x_k|z_{1:k-1})$$

Let's break this down using our vehicle tracking example. Imagine we have GPS measurements that give us position information. The measurement model $p(z_k|x_k)$ describes how our sensor readings relate to the true state, including sensor noise and limitations.

In its simplest form, for a linear system with Gaussian noise, the measurement model can be written as:

$$z(k) = H(k)x(k) + v(k)$$

Where: - $z(k)$ is the measurement vector - $H(k)$ is the measurement matrix that maps the state space to measurement space - $v(k)$ is the measurement noise (typically assumed Gaussian)

For our vehicle tracking example with GPS measurements, the measurement matrix $H(k)$ might look like:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This matrix indicates that we're only measuring position (x, y), not velocity or acceleration.

The actual update computation involves several steps:

1. First, we compute the innovation (the difference between predicted and actual measurements):
$$y(k) = z(k) - H(k)\hat{x}(k|k-1)$$

2. Then, we calculate the innovation covariance:
$$S(k) = H(k)P(k|k-1)H(k)^\top + R(k)$$
where $R(k)$ is the measurement noise covariance matrix

3. Next, we compute the Kalman gain, which determines how much we trust our measurement versus our prediction:
$$K(k) = P(k|k-1)H(k)^\top S(k)^{-1}$$

4. Finally, we update our state estimate and its covariance:
$$\hat{x}(k|k) = \hat{x}(k|k-1) + K(k)y(k)$$
$$P(k|k) = (I - K(k)H(k))P(k|k-1)$$

The Kalman gain $K(k)$ is particularly interesting because it acts as a weighting factor. When our measurements are very precise (small R), the Kalman gain will be

larger, meaning we trust the measurements more. When our measurements are noisy (large R), the gain will be smaller, and we'll trust our predictions more.

Consider what happens in extreme cases: - If our GPS suddenly becomes very accurate ($R \to 0$), K will increase, and we'll trust the GPS more - If our GPS is experiencing interference (large R), K will decrease, and we'll rely more on our motion model

This adaptive behavior is what makes recursive state estimation so powerful. The system automatically balances between prediction and measurement based on their relative uncertainties.

An important practical consideration is choosing appropriate values for the measurement noise covariance $R(k)$. This matrix represents our understanding of sensor characteristics and limitations. For a GPS sensor, we might set:

$$\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}$$

where σ_x and σ_y represent our uncertainty in x and y measurements.

3.4 Chapman-Kolmogorov equation

The Chapman-Kolmogorov equation is a fundamental concept in probability theory and plays a crucial role in state estimation. This equation describes how probability distributions evolve over time in a Markov process.

The Chapman-Kolmogorov equation is mathematically expressed as:

$$p(x_{k+1}|z_{1:k}) = \int p(x_{k+1}|x_k)p(x_k|z_{1:k})dx_k$$

Think of this equation as describing how we can "step forward" in time with our probability distributions. Let's break down what each part means and why it's important.

The left side, $p(x_{k+1}|z_{1:k})$, represents our prediction of the state at time k+1, given all measurements up to time k. This is what we want to calculate.

On the right side, we have two key components:

1. $p(x_{k+1}|x_k)$ is our motion model - how the state evolves from one time step to the next
2. $p(x_k|z_{1:k})$ is our current belief about the state at time k

The integral combines these components across all possible current states. It's like considering every possible current state, figuring out where it might lead, and weighting those possibilities by how likely we think each current state is.

Let's make this concrete with an example. Imagine tracking a car on a one-dimensional road:

- Current position is x
- Future position is x

- We have some uncertainty about both

The Chapman-Kolmogorov equation tells us to:

1. Consider each possible current position
2. For each current position, consider all possible future positions
3. Weight each possibility by how likely we think it is
4. Sum up all these weighted possibilities

This process naturally handles uncertainty propagation. If we're very uncertain about the current state, this uncertainty will be reflected in our prediction through the integration process.

The equation becomes particularly elegant when working with Gaussian distributions. In this case, if:

- Current belief is Gaussian with mean μ_k and variance σ_k^2
- Motion model is Gaussian with mean shift δ and variance τ^2

Then the prediction will also be Gaussian with:

- Mean: $\mu_{k+1} = \mu_k + \delta$
- Variance: $\sigma_{k+1}^2 = \sigma_k^2 + \tau^2$

This shows how uncertainties add up as we make predictions further into the future.

The Chapman-Kolmogorov equation is particularly important because:

1. It forms the theoretical foundation for the prediction step in Bayesian filtering
2. It respects the Markov property we discussed earlier
3. It provides a mathematically rigorous way to propagate uncertainties
4. It connects continuous and discrete-time processes

In practical applications, we often can't solve the integral analytically. This leads to various approximation methods:

- Kalman filters use Gaussian approximations
- Particle filters use numerical sampling
- Grid-based methods discretize the state space

3.5 Bayes filter algorithm

The Bayes filter is a probabilistic approach to estimating the state of a dynamic system over time using noisy measurements. Think of it as a mathematical framework for maintaining an educated guess about what's happening in a system, constantly refining that guess as new information arrives.

The core principle rests on maintaining a belief state - a probability distribution over all possible states. This belief state represents our uncertainty about the true state of the system. Let's break down how this works through the two main steps that occur recursively:

The Prediction Step (Time Update): In this first phase, we predict how our system will evolve based on our understanding of its dynamics. Imagine tracking a flying drone - even without looking at it, we can predict where it should be based on physics and our last known information about its position and velocity. This step uses the Chapman-Kolmogorov equation we discussed earlier:

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

This equation tells us to consider all possible previous states and how they might evolve into current states. The result is a prediction that accounts for all uncertainties in the system's dynamics.

The Correction Step (Measurement Update): When we get new sensor information, we update our prediction using Bayes' rule:

$$p(x_k|z_{1:k}) = \eta p(z_k|x_k)p(x_k|z_{1:k-1})$$

Here, we're weighing our prediction against new evidence, much like a detective updating their theory based on new clues.

Let's make this concrete with an example of a self-driving car:

Starting State:

- The car believes it's at a certain position with some uncertainty
- This belief is represented as a probability distribution over possible positions

Prediction Phase:

- The car knows it's moving forward at 30 mph
- Using this information and basic physics, it predicts where it should be after a small time interval
- The uncertainty grows during this prediction (the car might be sliding slightly, or its speedometer might be imperfect)

Measurement Phase:

- The car's GPS provides a new position reading
- The car's cameras detect lane markers
- These measurements are combined with the prediction to form an updated belief
- The uncertainty typically decreases during this phase as new evidence arrives

The beauty of the Bayes filter lies in how it handles uncertainty:

- If sensors are very accurate, their measurements are weighted more heavily
- If the motion is very predictable, the predictions are trusted more
- The algorithm automatically balances these factors based on their relative uncertainties

Real-world implementations often make specific assumptions about the nature of uncertainties and system dynamics. The most famous variant is the Kalman filter, which assumes:

- Linear system dynamics
- Gaussian uncertainties
- Additive noise

However, the general Bayes filter framework can handle non-linear systems and non-Gaussian uncertainties through variants like:

- Extended Kalman Filter: Handles mild non-linearities through linearization
- Unscented Kalman Filter: Better handles non-linear systems
- Particle Filter: Can handle any type of uncertainty or dynamics

3.6 Linear vs nonlinear systems

In the context of state estimation, a system's linearity or nonlinearity affects how states evolve over time and how measurements relate to states. This distinction is crucial because it determines which filtering approaches we can use effectively.

3.6.1 Linear Systems

A system is linear if it satisfies two key properties: superposition and homogeneity. In state estimation terms, this means:

For the motion model, a linear system follows the form:

$$x(k+1) = Ax(k) + Bu(k) + w(k)$$

Where:

- A is the state transition matrix
- B is the control input matrix
- w(k) is process noise
- All relationships between variables are strictly linear

For the measurement model, linearity means:

$$z(k) = Hx(k) + v(k)$$

Where:

- H is the measurement matrix
- v(k) is measurement noise

Consider tracking a train moving along a straight track at constant speed. This is approximately linear because:

- Position changes linearly with time
- Velocity remains constant
- Measurements (like position from track sensors) are directly proportional to state

3.6.2 Nonlinear Systems

Real-world systems are often nonlinear. The state evolution or measurements might involve:

- Trigonometric functions
- Quadratic terms
- Products of state variables
- Any other nonlinear mathematical relationships

The general form becomes:

$$x(k+1) = f(x(k), u(k)) + w(k)$$
$$z(k) = h(x(k)) + v(k)$$

Where f() and h() are nonlinear functions.

Consider tracking an aircraft:

- Position updates involve trigonometric functions of orientation
- Aerodynamic forces are quadratic with velocity
- Radar measurements give range and bearing, requiring nonlinear conversions to Cartesian coordinates

The implications for filtering are profound:

For Linear Systems:

- The Kalman Filter provides an optimal solution
- Uncertainties remain Gaussian if noise is Gaussian
- Computations are relatively simple and fast
- Results are guaranteed to converge under certain conditions

For Nonlinear Systems:

- The basic Kalman Filter no longer works optimally
- Uncertainties may become non-Gaussian even with Gaussian noise
- We need more sophisticated approaches:
  - Extended Kalman Filter (EKF): Linearizes around current estimate
  - Unscented Kalman Filter (UKF): Uses carefully chosen sample points
  - Particle Filter: Represents uncertainty with discrete particles

Let's consider a specific example: a pendulum.

- Linear approximation works when swing angle is small (sin( ))
- As swing amplitude increases, nonlinear effects become important
- The true motion involves trigonometric functions of angle

This demonstrates how real systems might be approximated as linear within certain operating ranges but require nonlinear treatment for full accuracy.

Chapter 4

Kalman Filtering

- ☐ Kalman Filtering
  - ☐ Linear Kalman Filter
    - ☐ System model and assumptions
    - ☐ Prediction equations
    - ☐ Update equations
    - ☐ Uncertainty propagation
  - ☐ Extended Kalman Filter (EKF)
    - ☐ Linearization process
    - ☐ Jacobian matrices
    - ☐ Algorithm implementation
  - ☐ Unscented Kalman Filter (UKF)
    - ☐ Sigma points
    - ☐ Unscented transform
    - ☐ Algorithm implementation

Chapter 5

Particle Filtering

- ☐ Particle Filtering
  - ☐ Monte Carlo methods
  - ☐ Importance sampling
  - ☐ Particle representation
  - ☐ Sequential Importance Sampling (SIS)
  - ☐ Resampling techniques
  - ☐ Sample degeneracy and impoverishment
  - ☐ Adaptive particle filtering

Chapter 6

Advanced Topics

- ☐ Advanced Topics
  - ☐ Multi-hypothesis tracking
  - ☐ SLAM basics
  - ☐ Sensor fusion techniques
  - ☐ Loop closure
  - ☐ Global vs local localization

Chapter 7

MATLAB Implementation: Kalman Filter

- ☐ MATLAB Implementation: Kalman Filter
  - ☐ Linear KF implementation
    - ☐ State prediction
    - ☐ Measurement update
    - ☐ Covariance propagation
  - ☐ EKF implementation
    - ☐ System modeling
    - ☐ Jacobian computation
    - ☐ Filter implementation
  - ☐ Visualization and analysis
  - ☐ Performance evaluation

Chapter 8

MATLAB Implementation: Particle Filter

- ☐ MATLAB Implementation: Particle Filter
  - ☐ Basic PF framework
  - ☐ Particle initialization
  - ☐ Motion model implementation
  - ☐ Measurement model
  - ☐ Weight computation
  - ☐ Resampling implementation
  - ☐ Visualization tools
  - ☐ Performance metrics

Chapter 9

Practical Applications

- ☐ Practical Applications
  - ☐ Vehicle localization case studies
  - ☐ Robot navigation examples
  - ☐ Integration with mapping
  - ☐ Real-world challenges
  - ☐ Best practices and optimization

Chapter 10

Project Work

- ☐ Project Work
  - ☐ Implementation exercises
  - ☐ Real dataset analysis
  - ☐ Performance comparison of different filters
  - ☐ Parameter tuning
  - ☐ Documentation and presentation