Localization & Mapping

Today’s Agenda

• 400-level Requirements
• Homework 4 Preview
• Localization I
Localization and Map Building

- Noise and aliasing; odometric position estimation
- To localize or not to localize
- Belief representation
- Map representation
- Probabilistic map-based localization
- Other examples of localization systems
- Autonomous map building

Localization, Where am I?

- Odometry, Dead Reckoning
- Localization based on external sensors, beacons or landmarks
- Probabilistic Map Based Localization
Challenges of Localization

• Knowing the absolute position (e.g. GPS) is not sufficient
• Localization in human-scale in relation with environment
• Planing in the Cognition step requires more than only position as input
• Perception and motion plays an important role
  ➢ Sensor noise
  ➢ Sensor aliasing
  ➢ Effector noise
  ➢ Odometric position estimation

Sensor Noise

• Sensor noise in mainly influenced by environment e.g. surface, illumination …
• AND by the measurement principle itself e.g. interference between ultrasonic sensors
• Sensor noise drastically reduces the useful information of sensor readings. The solution is:
  ➢ to take multiple reading into account
  ➢ employ temporal and/or multi-sensor fusion
Sensor Aliasing

- In robots, non-uniqueness of sensors readings is the norm
- Even with multiple sensors, there is a many-to-one mapping from environmental states to robot’s perceptual inputs
- Therefore the amount of information perceived by the sensors is generally insufficient to identify the robot’s position from a single reading
  - Robot’s localization is usually based on a series of readings
  - Sufficient information is recovered by the robot over time

Effector Noise: Odometry, Dead Reckoning

- Odometry and dead reckoning:
  - Position update is based on proprioceptive sensors
  - Odometry: wheel sensors only
  - Dead reckoning: also heading sensors
- The movement of the robot, sensed with wheel encoders and/or heading sensors is integrated to the position.
  - Pros: Straight forward, easy
  - Cons: Errors are integrated -> unbound
- Using additional heading sensors (e.g. gyroscope) might help to reduce the cumulated errors, but the main problems remain the same.
Odometry: Error sources

<table>
<thead>
<tr>
<th>deterministic</th>
<th>non-deterministic</th>
</tr>
</thead>
<tbody>
<tr>
<td>(systematic)</td>
<td>(non-systematic)</td>
</tr>
</tbody>
</table>

• Deterministic errors can be eliminated by proper calibration of the system.
• Non-deterministic errors have to be described by error models and will always lead to uncertain position estimate.

• Major Error Sources:
  - Limited resolution during integration (time increments, measurement resolution, etc.)
  - Misalignment of the wheels (deterministic)
  - Unequal wheel diameter (deterministic)
  - Variation in the contact point of the wheel
  - Unequal floor contact (slipping, not planar …), etc.

Odometry: Classification of Integration Errors

• Range error: integrated path length (distance) of the robots movement
  - Sum of the wheel movements

• Turn error: similar to range error, but for turns
  - Difference of the wheel motions

• Drift error: difference in the error of the wheels leads to an error in the robots angular orientation

  Over long periods of time, turn and drift errors far outweigh range errors!

  Consider moving forward on a straight line along the x axis. The error in the y-position introduced by a move of d meters will have a component of \(d \sin \Delta \theta\), which can be quite large as the angular error \(\Delta \theta\) grows.
Odometry: The Differential Drive Robot (1)

\[
p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \quad p' = p + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix}
\]

Odometry: The Differential Drive Robot (2)

• Kinematics

\[
\begin{align*}
\Delta x &= \Delta s \cos(\theta + \Delta \theta/2) \\
\Delta y &= \Delta s \sin(\theta + \Delta \theta/2) \\
\Delta \theta &= \frac{\Delta s_r - \Delta s_l}{b} \\
\Delta s &= \frac{\Delta s_r + \Delta s_l}{2} \\
p' &= f(x, y, \theta, \Delta s_r, \Delta s_l) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta s_r + \Delta s_l \cos\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\
\Delta s_r + \Delta s_l \sin\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\
\frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix}
\end{align*}
\]
STAT Refresher

• The expected value for a random variable $X$ is (i.e. the mean) defined as

$$\mu = E(X) = \sum_{i=1}^{n} p_i x_i \text{ for discrete } X$$

$$\mu = E(X) = \int_{-\infty}^{\infty} x p_X(x) dx \text{ for continuous } X$$

• The variance of $X$ about the mean is defined as

$$\sigma^2 = E[(x - \mu)^2] = \sum_{i=1}^{n} p_i (x_i - \mu)^2 \text{ for discrete } X$$

$$\sigma^2 = E[(x - \mu)^2] = \int_{-\infty}^{\infty} (x - \mu)^2 p_X(x) dx \text{ for continuous } X$$

STAT Refresher (2)

• When $X$ is a vector, the variance is expressed in terms of a covariance matrix $C$ where

$$c_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)]$$

• The resulting matrix has the form

$$C = \begin{bmatrix}
\sigma_1^2 & \rho_{12}\sigma_1\sigma_2 & \cdots & \rho_{1n}\sigma_1\sigma_n \\
\rho_{12}\sigma_1\sigma_2 & \sigma_2^2 & \cdots & \rho_{2n}\sigma_2\sigma_n \\
\vdots & \vdots & \ddots & \vdots \\
\rho_{1n}\sigma_1\sigma_n & \rho_{2n}\sigma_2\sigma_n & \cdots & \sigma_n^2
\end{bmatrix}$$

where $\rho_{ij}$ corresponds to the degree of correlation between variables $X_i$ and $X_j$
Correlation is a means to estimate how two functions/series are correlated. For a discrete series, it is defined as

$$\rho = \frac{\sum_i [(x_i - \mu_x)(y_i - \mu_y)]}{\sqrt{\sum_i (x_i - \mu_x)^2} \sqrt{\sum_i (y_i - \mu_y)^2}} = \frac{C_{xy}}{\sigma_x \sigma_y}$$

where $\rho$ denotes the correlation coefficient

- The denominator normalizes the correlation coefficient such that

$$\rho \in [-1, 1]$$
Correlation Examples

\[ \rho = 0.97 \]

Correlation Examples

\[ \rho = 0.31 \]
Properties of the Covariance Matrix

- The covariance matrix $C$ is symmetric and positive definite.
- Through a similarity transform with the proper rotation matrix $R$, $C$ can be decomposed as $C = RDR^T$, where

$$C = \begin{bmatrix} \text{EVE}_1 & \cdots & \text{EVE}_n \end{bmatrix} \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_n^2 \end{bmatrix} \begin{bmatrix} \text{EVE}_1 \\ \vdots \\ \text{EVE}_n \end{bmatrix}$$

- That is, the columns of $R$ correspond to the eigenvectors of $C$, and the elements of the diagonal matrix $D$ its eigenvalues.
- These also correspond to the primary axes of the PDF and the variances, respectively.
- This means you can always define a coordinate system where the values will be uncorrelated!

Example

- QUESTION: Given a normal distribution with covariance matrix

$$C = \begin{bmatrix} 5.4729 & 3.9719 \\ 3.9719 & 4.5271 \end{bmatrix}$$

characterize the PDF.

- SOLUTION: Solving for the eigenvalues, we get

$$(\lambda - 5.47)(\lambda - 4.53) - 3.97^2 = 0$$

$\Rightarrow \lambda_1 = 9, \lambda_2 = 1 \Rightarrow \sigma_1^2 = 9, \sigma_2^2 = 1$

and solving for the first EVE we obtain

$$x_1 = \begin{bmatrix} 0.747 \\ 0.664 \end{bmatrix} \Rightarrow \tan^{-1}\left( \frac{0.664}{0.747} \right) = 41.6^\circ$$

What this means for us is that we can always extract a nice 2D ellipse to reflect our positional uncertainty on the plane.
The Gaussian Noise Assumption

- A 1-D Gaussian distribution is defined as
  \[ p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \]

- In 2-D (assuming uncorrelated variables) this becomes
  \[ p(x) = \frac{1}{2\pi\sigma_1\sigma_2} e^{-\frac{(x_1-\mu_1)^2}{2\sigma_1^2} - \frac{(x_2-\mu_2)^2}{2\sigma_2^2}} \]

- In \( n \) dimensions, it generalizes to
  \[ p(x) = \frac{1}{\sqrt{(2\pi)^n |C|}} e^{-\frac{1}{2} (x-\mu)^T C^{-1} (x-\mu)} \]

The Normal (Gaussian) distribution is completely parameterized by its first and second moments.

Properties of Gaussians...

\( X \sim N(\mu_x, \sigma_x^2) \quad Y \sim N(\mu_y, \sigma_y^2) \)

- Given two independent random variables \( X, Y \) with respective normal distributions, then
  \[ aX + b \sim N(a\mu_x + b, a^2\sigma_x^2) \]
  \[ Z = X + Y \sim N(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2) \]

- However, in many cases when we propagate the covariance, the underlying function needs is \textit{NOT} linear
- Q: How do you think we could do this?
- A: Linearization.
- Again, our wonderful friend the Taylor series comes to the rescue
  \[ f(x + \epsilon) = f(x) + f'(x)\epsilon + \ldots \]
  \[ f(x + \epsilon) = f(\bar{x}) + J\epsilon + \ldots \]
**Transforming Uncertainty (1)**

- Let’s say we know the uncertainty of a variable $x$, and we want to compute the uncertainty of $y = f(x)$
- We know that $\bar{x} = \bar{x} + \varepsilon$
  - where $\bar{x}$ is the distribution mean and $\varepsilon$ is zero mean noise
- We can then use the Jacobian to linearly approximate $y$
  \[
  \bar{y} = f(\bar{x}) = f(\bar{x} + \varepsilon) \approx f(\bar{x}) + J\varepsilon
  \]
- The mean of the distribution would then be
  \[
  \bar{y} = E[\bar{y}] = E[f(\bar{x}) + J\varepsilon] = f(\bar{x})
  \]
- Therefore $\bar{y} - \bar{y} \approx J\varepsilon$

**Transforming Uncertainty (2)**

- The covariance of the transformed distribution would then be
  \[
  C_y = E[(\bar{y} - \bar{y})(\bar{y} - \bar{y})^T] \approx E[J\varepsilon\varepsilon^T J^T] = JC_x J^T
  \]
- Thus, to transform uncertainty across a non-linear transformation, we perform a similarity transform with the Jacobian
- Note that because of the $\approx$ symbols on the previous page, normal distributions are NOT preserved
Odometry: The Differential Drive Robot (3)

• Error model

\[ \Sigma_\Delta = covar(\Delta s_x, \Delta s_y) = \begin{bmatrix} k_f |\Delta s_x| & 0 \\ 0 & k_f |\Delta s_y| \end{bmatrix} \]

\[ \Sigma_{\Delta f} = \nabla \rho \cdot f \cdot \Sigma_{\Delta f} \cdot f^T + \nabla \Delta \rho \cdot f \cdot \Sigma_{\Delta f} \cdot f^T \]

\[ F_{\Delta f} - \nabla \rho \cdot f = \begin{bmatrix} \frac{d}{dx} \Delta \rho, \frac{d}{dy} \Delta \rho \end{bmatrix} = \begin{bmatrix} 1 & -\Delta \sin(\theta + \Delta \theta/2) \\ -\Delta \sin(\theta + \Delta \theta/2) & 1 \end{bmatrix} \]

\[ F_{\Delta x} = \begin{bmatrix} 1 \cos(\theta + \Delta \theta/2) \Delta \rho \sin(\theta + \Delta \theta/2) \\ \sin(\theta + \Delta \theta/2) \Delta \rho \cos(\theta + \Delta \theta/2) \end{bmatrix} \]

Odometry: Growth of Pose uncertainty for Straight Line Movement

• Note: Errors perpendicular to the direction of movement are growing much faster!
Odometry: Growth of Pose uncertainty for Movement on a Circle

- Note: Errors ellipse in does not remain perpendicular to the direction of movement!

To Localize or Not to Localize?

- How to navigate between A and B
  - navigation without hitting obstacles
  - detection of goal location
- Possible by following always the left wall
  - However, how to detect that the goal is reached
Behavior Based Navigation

- communicate data
- discover new area
- detect goal position
- avoid obstacles
- follow right / left wall

Actuators

Coordination / fusion
E.g. fusion via vector summation

Model Based Navigation

- perception
- localization / map-building
- cognition / planning
- motion control

Actuators
Belief Representation

• a) Continuous map with single hypothesis

• b) Continuous map with multiple hypothesis

• c) Discretized map with probability distribution

• d) Discretized topological map with probability distribution

Belief Representation: Characteristics

• Continuous
  ➢ Precision bound by sensor data
  ➢ Single or multiple hypothesis pose estimate
  ➢ Lost when diverging (for single hypothesis)
  ➢ Compact representation and typically reasonable in processing power.

• Discrete
  ➢ Precision bound by resolution of discretisation
  ➢ Typically multiple hypothesis pose estimate
  ➢ Never lost (when diverges converges to another cell)
  ➢ Important memory and processing power needed. (not the case for topological maps)
Single-hypothesis Belief – Continuous Line-Map

a) Robot position

Single-hypothesis Belief – Grid and Topological Map

c) Grid map

d) Topological map

© R. Siegwart, I. Nourbakhsh
Grid-base Representation - Multi Hypothesis

- Grid size around 20 cm$^2$.

Planning and Computational Issues!

Multi-Hypothesis Issues

- Can explicitly maintain uncertainty estimate regarding the robot’s position
- Allows the use of partial information
- However:
  - Where is the robot?
  - How do you handle path planning?
Map Representation

- “Dual” of representing the robot’s possible position(s)
- Effects the choices available for robot position representation
- Three Fundamental questions:
  1. Map precision vs. application
  2. Features precision vs. map precision
  3. Precision vs. computational complexity

Continuous Representation
- Exact decomposition of the environment

Discretization

Representation of the Environment

- Environment Representation
  - Continuous Metric \(\rightarrow x, y, \theta\)
  - Discrete Metric \(\rightarrow\) metric grid
  - Discrete Topological \(\rightarrow\) topological grid

- Environment Modeling
  - Raw sensor data, e.g. laser range data, grayscale images
    - large volume of data, low distinctiveness on the level of individual values
    - makes use of all acquired information
  - Low level features, e.g. line other geometric features
    - medium volume of data, average distinctiveness
    - filters out the useful information, still ambiguities
  - High level features, e.g. doors, a car, the Eiffel tower
    - low volume of data, high distinctiveness
    - filters out the useful information, few/no ambiguities, not enough information
**5.5.1** Map Representation: Continuous Line-Based

- a) Architecture map
- b) Representation with set of infinite lines

*Key advantage is potentially high accuracy.*

**5.5.2** Map Representation: Decomposition (1)

- Exact cell decomposition
Decomposition Strategies

- Abstraction
- The underlying assumption is that the exact position of a robot within each area of free space does not matter
- Tessellate the environment into “useful” regions
- Loss of map fidelity can sometimes be apparent

Map Representation: Decomposition (2)

- Fixed cell decomposition
  - Narrow passages disappear
Map Representation: Decomposition (3)

- Adaptive cell decomposition

Occupancy Grid

- Environment represented by a discrete grid
- Each cell classified as either “free” or “occupied” by an obstacle
- Each cell has either a “counter” or a PDF associated with its likelihood of being an obstacle or open
- A threshold on the number of hits/probability is used for deciding what is/is not an obstacle
- If range measurements pass through cell, counter/probability discounted
- Little Ben Video
Map Representation: Decomposition (4)

- Fixed cell decomposition – Example with very small cells

Map Representation: Decomposition (5)

- Topological Decomposition
Topological Maps

- Graph representation
- Nodes represent areas in the world
- Edges represent connectivity between adjacent areas
- Demarcation of areas/transitions is tuned to sensor capabilities
  - Red room
  - Intersection detector
  - Doorway detector
- An advantage of the topological representation is that it embeds non-geometric information
  - Color
  - RFID

Map Representation: Decomposition (6)

- Topological Decomposition

![Diagram of a topological map with nodes and arcs representing connectivity.]
Map Representation: Decomposition (7)

- Topological Decomposition

A limitation is that sensors are often tuned to the environment and do not translate well from one to another.

Current Challenges in Map Representation

- Real world is dynamic
- Perception is still a major challenge
  - Error prone
  - Extraction of useful information difficult
- Traversal of open space
- How to build up topology (boundaries of nodes)
- Sensor fusion
- …
Probabilistic, Map-Based Localization (1)

• Consider a mobile robot moving in a known environment.

• As it starts to move, say from a precisely known location, it might keep track of its location using odometry.

• However, after a certain movement the robot will get very uncertain about its position.

• update using an observation of its environment.

• Observations also yield an estimate of the robot’s position which can then be fused with the odometric estimation to get the best possible update of the robot’s actual position.

Bayes Law Review

• \( p(A \mid B) \): The probability of \( A \) given \( B \) has occurred (the conditional probability of \( A \) on \( B \))

• Theorem of Compound Probability:

\[
p(A \land B) = p(A \cap B) = p(B \mid A) p(A)
\]

\[
p(A \land B) = p(A \cap B) = p(A \mid B) p(B)
\]

• Bayes Law

\[
p(B)p(A \mid B) = p(A)p(B \mid A)
\]

\[
\Rightarrow \quad p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B)}
\]
A Simple Example (1)

- Suppose our robot is trying to detect obstacles from a measurement $s$.
- What is $p(\text{obstacle}|s)$?

A Simple Example (2)

- $p(\text{obstacle}|s)$ in practice is difficult to measure explicitly.
- Since we have a sensor model, it is often easier to get $p(s|\text{obstacle})$.
- By applying Bayes law we obtain

$$p(\text{obst} \mid s) = \frac{p(s \mid \text{obst}) p(\text{obst})}{p(s)}$$
Normalization Process

\[
p(\text{obst} \mid s) = \frac{p(s \mid \text{obst}) p(\text{obst})}{p(s)}
\]

\[
p(\neg \text{obst} \mid s) = \frac{p(s \mid \neg \text{obst}) p(\neg \text{obst})}{p(s)}
\]

\[
p(\text{obst} \mid s) + p(\neg \text{obst} \mid s) = 1
\]

\[
\Rightarrow p(s) = p(s \mid \text{obst}) p(\text{obst}) + p(s \mid \neg \text{obst}) p(\neg \text{obst})
\]

\[
p(\text{obst} \mid s) = \frac{p(s \mid \text{obst}) p(\text{obst})}{p(s \mid \text{obst}) p(\text{obst}) + p(s \mid \neg \text{obst}) p(\neg \text{obst})}
\]

A Simple Example (3)

- Suppose our robot is trying to detect obstacles from a measurement \( s \)
- What is \( p(\text{obstacle} \mid \text{detection}) \) \( IF \)
  - \( p(\text{obstacle}) = 0.1 \)
  - \( p(\text{detection} \mid \text{obstacle}) = 0.9 \)
  - \( p(\text{detection} \mid \neg \text{obstacle}) = 0.05 \)

\[
p(o \mid s) = \frac{0.9 \times 0.1}{0.9 \times 0.1 + 0.05 \times 0.9} = \frac{2}{3}
\]
What if Additional Information is Available?

- With additional information $C$ available, Bayes Theorem becomes
  \[
  p(A \mid B_1, B_2) = \frac{p(B_2 \mid A, B_1) p(A \mid B_1)}{p(B_2 \mid B_1)}
  \]

- The recursive formula for Bayesian Updating is then
  \[
  p(A \mid B_1, \ldots, B_n) = \frac{p(B_n \mid A, B_1, \ldots, B_{n-1}) p(A \mid B_1, \ldots, B_{n-1})}{p(B_n \mid B_1, \ldots, B_{n-1})}
  \]

- If $B_n$ is independent from $B_1, \ldots, B_{n-1}$ GIVEN $A$ this reduces to
  \[
  p(A \mid B_1, \ldots, B_n) = \frac{p(B_n \mid A) p(A \mid B_1, \ldots, B_{n-1})}{p(B_n \mid B_1, \ldots, B_{n-1})}
  \]

A Simple Example (4)

- Suppose our robot has a second detector $s_2$
  
  - $p(detection \mid obstacle) = 0.5$
  - $p(detection \mid no \ obstacle) = 0.05$

\[
\begin{align*}
p(o \mid s_2) &= \frac{0.5 \times 0.1}{0.5 \times 0.1 + 0.05 \times 0.9} = \frac{10}{19}, \quad p(o \mid s_1) = \frac{2}{3} \\
p(o \mid s_1, s_2) &= \frac{p(s_2 \mid o, s_1) p(o \mid s_1)}{p(s_2 \mid s_1)} \\
&= \frac{p(s_2 \mid o) p(o \mid s_1)}{p(s_2 \mid o) p(o \mid s_1) + p(s_2 \mid !o) p(!o \mid s_1)} = \frac{0.5 \times \frac{2}{3}}{0.5 \times \frac{2}{3} + 0.05 \times \frac{1}{3}} = 0.95
\end{align*}
\]
Suppose our robot has a second detector $s_2$

- $p(\text{detection}|\text{obstacle}) = 0.5$
- $p(\text{detection}|\text{no obstacle}) = 0.05$

\[
p(o|s_2) = \frac{0.5 \times 0.1}{0.5 \times 0.1 + 0.05 \times 0.9} = \frac{10}{19}, \quad p(o|s_1) = \frac{2}{3}
\]

\[
p(o|s_1, s_2) = \frac{p(s_1|o, s_2) p(o|s_2)}{p(s_1|s_2)} = \frac{p(s_1|o) p(o|s_2)}{p(s_1|o)p(o|s_2) + p(s_1|\neg o)p(\neg o|s_2)} = \frac{0.9 \times \frac{10}{19}}{0.9 \times \frac{10}{19} + 0.05 \times \frac{9}{19}} = 0.95
\]

**Proportional, Map-Based Localization (2)**

- **Action update**
  - action model $ACT$
  
  \[
  s'_t = Act(o_t, s_{t-1})
  \]
  
  with $o_t$: Encoder Measurement, $s_{t-1}$: prior belief state
  
  - increases uncertainty

- **Perception update**
  - perception model $SEE$
  
  \[
  s_t = See(i_t, s'_t)
  \]
  
  with $i_t$: exteroceptive sensor inputs, $s'_t$: updated belief state
  
  - decreases uncertainty
• Improving belief state by moving

The Five Steps for Map-Based Localization

1. Prediction based on previous estimate and odometry
2. Observation with on-board sensors
3. Measurement prediction based on prediction and map
4. Matching of observation and map
5. Estimation → position update (posterior position)
Markov \iff\ Kalman Filter Localization

- **Markov localization**
  - localization starting from any unknown position
  - recovers from ambiguous situation.
  - However, to update the probability of all positions within the whole state space at any time requires a discrete representation of the space (grid). The required memory and calculation power can thus become very important if a fine grid is used.

- **Kalman filter localization**
  - tracks the robot and is inherently very precise and efficient.
  - However, if the uncertainty of the robot becomes too large (e.g., collision with an object) the Kalman filter will fail and the position is definitively lost.

Markov Localization (1)

- Markov localization uses an explicit, discrete representation for the probability of all positions in the state space.

- This is usually done by representing the environment by a grid or a topological graph with a finite number of possible states (positions).

- During each update, the probability for each state (element) of the entire space is updated.
Markov Localization (2): Applying probability theory to robot localization

- $P(A)$: Probability that $A$ is true.
  - e.g. $p(r_t = l)$: probability that the robot $r$ is at location $l$ at time $t$
- We wish to compute the probability of each individual robot position given actions and sensor measures.
- $P(A|B)$: Conditional probability of $A$ given that we know $B$.
  - e.g. $p(r_t = l | i_t)$: probability that the robot is at position $l$ given the sensors input $i_t$.
- Product rule:
  \[ p(A \land B) = p(A|B)p(B) \]
  \[ p(A \land B) = p(B|A)p(A) \]
- Bayes rule:
  \[ p(A|B) = \frac{p(B|A)p(A)}{p(B)} \]

Markov Localization (3): Applying probability theory to robot localization

- Bayes rule:
  \[ p(A|B) = \frac{p(B|A)p(A)}{p(B)} \]
  
  - Map from a belief state and a sensor input to a refined belief state (SEE):
    \[ p(l|i) = \frac{p(i|l)p(l)}{p(i)} \]
  
  - $p(l)$: belief state before perceptual update process
  - $p(i|l)$: probability to get measurement $i$ when being at position $l$
    - consult robots map, identify the probability of a certain sensor reading for each possible position in the map
  - $p(i)$: normalization factor so that sum over all $l$ for $L$ equals 1.
Markov Localization (4): Applying probability theory to robot localization

- Bayes rule:
  \[ p(A|B) = \frac{p(B|A)p(A)}{p(B)} \]

- Map from a belief state and an action to new belief state (ACT):
  \[ p(l_i|o_t) = \int p(l_i|l_{t-1}^n, o_t)p(l_{t-1}^n)dl_{t-1}^n \]

- Summing over all possible ways in which the robot may have reached \( l \).

- Markov assumption: Update only depends on previous state and its most recent actions and perception.

---

Markov Localization: Case Study 1 - Topological Map (1)

- The Dervish Robot
- Topological Localization with Sonar
Markov Localization: Case Study 1 - Topological Map (2)

- Topological map of office-type environment

<table>
<thead>
<tr>
<th></th>
<th>Wall</th>
<th>Closed door</th>
<th>Open door</th>
<th>Open hallway</th>
<th>Foyer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nothing detected</td>
<td>0.70</td>
<td>0.40</td>
<td>0.05</td>
<td>0.001</td>
<td>0.30</td>
</tr>
<tr>
<td>Closed door detected</td>
<td>0.30</td>
<td>0.60</td>
<td>0</td>
<td>0</td>
<td>0.05</td>
</tr>
<tr>
<td>Open door detected</td>
<td>0</td>
<td>0</td>
<td>0.90</td>
<td>0.10</td>
<td>0.15</td>
</tr>
<tr>
<td>Open hallway detected</td>
<td>0</td>
<td>0</td>
<td>0.001</td>
<td>0.90</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Markov Localization: Case Study 1 - Topological Map (3)

- Update of believe state for position $n$ given the percept-pair $i$
  \[
P(n | i) = p(i | n)p(n)
  \]
  - $p(n | i)$: new likelihood for being in position $n$
  - $p(n)$: current believe state
  - $p(i | n)$: probability of seeing $i$ in $n$ (see table)

- No action update!
  - However, the robot is moving and therefore we can apply a combination of action and perception update
  \[
P(n_r | i_t) = \int p(n_r | n_{r-1}, i_t)p(n_{r-1} | i_t)dn_{r-1}
  \]
  - $t-i$ is used instead of $t-1$ because the topological distance between $n'$ and $n$ can vary depending on the specific topological map
Markov Localization: Case Study 1 - Topological Map (4)

- The calculation
  \[ p(n_i | n_{i-1}', i) \]
  is calculated by multiplying the probability of generating perceptual event \( i \) at position \( n \) by the probability of having failed to generate perceptual events at all nodes between \( n' \) and \( n \).

\[ p(n_i | n_{i-1}', i) = p(i, n_i) \cdot p(\emptyset, n_{i-1}) \cdot p(\emptyset, n_{i-2}) \cdot \ldots \cdot p(\emptyset, n_{i-1} + 1) \]

Example calculation

- Assume that the robot has two nonzero belief states
  - \( p(1-2) = 1.0 \)
  - \( p(2-3) = 0.2 \)

  State 2-3 will progress potentially to 3 and 3-4 to 4
  State 3 and 3-4 can be eliminated because the likelihood of detecting an open door is zero.

- The likelihood of reaching state 4 is the product of the initial likelihood \( p(2-3) = 0.2 \), (a) the likelihood of detecting anything at node 3 and the likelihood of detecting a hallway on the left and a door on the right at node 4 and (b) the likelihood of detecting a hallway on the left and a door on the right at node 4. (for simplicity we assume that the likelihood of detecting nothing at node 3-4 is 1.0)

- This leads to:
  - \( 0.2 \cdot (0.6 \cdot 0.4 + 0.4 \cdot 0.05) \cdot 0.7 \cdot 0.9 \cdot 0.1 \) \( \rightarrow \) \( p(4) = 0.003 \).
  - Similar calculation for progress from 1-2 \( \rightarrow p(2) = 0.3 \).

* Note that the probabilities do not sum up to one. For simplicity normalization was avoided in this example.
Markov Localization: Case Study 2 – Grid Map (1)

- Fine fixed decomposition grid \((x, y, \theta)\), 15 cm x 15 cm x 1°
  - Action and perception update
  - Action update:
    - Sum over previous possible positions and motion model
    \[
P(l_t | o_t) = \sum_{l_{t-1}} P(l_t | l_{t-1}, o_t) \cdot p(l_{t-1})
    \]
    - Discrete version of eq. 5.22
  - Perception update:
    - Given perception \(i\), what is the probability to be a location \(l\)
    \[
p(l | i) = \frac{p(i | l)p(l)}{p(i)}
    \]

Markov Localization: Case Study 2 – Grid Map (2)

- The critical challenge is the calculation of \(p(i | l)\)
  - The number of possible sensor readings and geometric contexts is extremely large
  - \(p(i | l)\) is computed using a model of the robot’s sensor behavior, its position \(l\), and the local environment metric map around \(l\).
  - Assumptions
    - Measurement error can be described by a distribution with a mean
    - Non-zero chance for any measurement

Courtesy of W. Burgard
Markov Localization: Case Study 2 – Grid Map (3)

• The 1D case

1. **Start**
   - No knowledge at start, thus we have an uniform probability distribution.

2. **Robot perceives first pillar**
   - Seeing only one pillar, the probability being at pillar 1, 2 or 3 is equal.

3. **Robot moves**
   - Action model enables to estimate the new probability distribution based on the previous one and the motion.

4. **Robot perceives second pillar**
   - Based on all prior knowledge the probability being at pillar 2 becomes dominant.

Markov Localization: Case Study 2 – Grid Map (4)

• Example 1: Office Building

    © R. Siegwart, I. Nourbakhsh

    Courtesy of W. Burgard
Markov Localization: Case Study 2 – Grid Map (5)

- Example 2: Museum
  - Laser scan 1

Markov Localization: Case Study 2 – Grid Map (6)

- Example 2: Museum
  - Laser scan 2
Markov Localization: Case Study 2 – Grid Map (7)

• Example 2: Museum
  ➢ Laser scan 3

Markov Localization: Case Study 2 – Grid Map (8)

• Example 2: Museum
  ➢ Laser scan 13
Markov Localization: Case Study 2 – Grid Map (9)

- Example 2: Museum
  - Laser scan 21

Markov Localization: Case Study 2 – Grid Map (10)

- Fine fixed decomposition grids result in a huge state space
  - Very important processing power needed
  - Large memory requirement

- Reducing complexity
  - Various approached have been proposed for reducing complexity
  - The main goal is to reduce the number of states that are updated in each step

- Randomized Sampling / Particle Filter
  - Approximated belief state by representing only a ‘representative’ subset of all states (possible locations)
  - E.g update only 10% of all possible locations
  - The sampling process is typically weighted, e.g. put more samples around the local peaks in the probability density function
  - However, you have to ensure some less likely locations are still tracked, otherwise the robot might get lost
The robot knows its position to be either A or B. It measures the distance to the wall in front of it to be 22 meters. Which position is the robot located?

What if there is a third possible position C?
What is a Particle Filter? (1)

- The uniform distribution is defined as:
  \[ p(x) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a,b] \\ 0 & \text{otherwise} \end{cases} \]

- Our friend, the Gaussian distribution:
  \[ p(x) = \frac{1}{2\pi \sigma_1 \sigma_2} e^{-\frac{(x_1-\mu_1)^2}{2\sigma_1^2} - \frac{(x_2-\mu_2)^2}{2\sigma_2^2}} \]

- Particle distributions

What is a Particle Filter? (2)

- Particle Filters (PF) are known by a variety of names (Importance Sampling, the Metropolis Algorithm, Monte Carlo Methods, CONDENSATION algorithm, etc.)

- We will use the PF moniker from statistical literature

- PFs represent a probability density function as a set of discrete samples or particles, each representing a hypothetical estimate of the state.

- For example, to estimate the pose of a robot on the plane, each particle would correspond to a hypothetical position and orientation.

- We can then take statistics over this set as necessary to estimate pose.
Bayesian Filters (1)

- PFs and Kalman Filters (KF/EKF) are example of Bayesian Filters
- Bayesian filters do not explicitly estimate the state
- Instead, they propagate a posterior probability density function for the state from which it can be inferred
- In the KF, a gaussian distribution $P$ is propagated at each timestep with mean $\mu$ and variance $\sigma^2$. The former is used as the state estimate
- In the PF, a (weighted) particle set corresponds to the posterior from which an estimate for the state can be inferred

Bayesian Filters (2)

- Let $x_t$ denote the state at time $t$
- Bayesian filters estimate the conditional pdf for $x_t$ or the belief denoted by

$$Bel(\tilde{x}_t) = p(\tilde{x}_t \mid d_{0..t})$$

where $d_{0..t}$ corresponds to all of the available data that the pdf has been conditioned upon
- For robot localization, $d_{0..t}$ would correspond to sensor measurements and robot motions. Denoting these as $z$ and $u$, respectively we obtain

$$Bel(\tilde{x}_t) = p(\tilde{x}_t \mid z_t, u_t, z_{t-1}, u_{t-1}, \ldots, z_0, u_0)$$

where the belief is conditioned upon all available sensor measurements and robot actions
Bayesian Filters (3)

- Bayesian filters use predictive corrective models. This recursive nature can be obtained by expanding the previous equation via Bayes rule

\[
Bel(\tilde{x}_t) = \frac{p(z_t \mid x_t, u_t, z_{t-1}, u_{t-1}, \ldots, u_0) p(\tilde{x}_t \mid u_t, z_{t-1}, u_{t-1}, \ldots, u_0)}{p(z_t \mid z_{t-1}, u_{t-1}, \ldots, u_0)}
\]

- Since the denominator is not a function of \(x_t\) and serves to normalize the distribution, we can replace it instead with a normalization constant \(\eta\)

\[
Bel(\tilde{x}_t) = \eta p(z_t \mid x_t, u_t, z_{t-1}, u_{t-1}, \ldots, u_0) p(\tilde{x}_t \mid u_t, z_{t-1}, u_{t-1}, \ldots, u_0)
\]

Bayesian Filters (4)

- Bayes filters than rely upon a Markov assumption which effectively means that our belief is not dependent on past sensor data or robot motions if one knows the current state. Mathematically, this implies that

\[
p(z_t \mid x_t, u_t, z_{t-1}, u_{t-1}, \ldots, u_0) = p(z_t \mid x_t)
\]

- With this assumption, our previous equation reduces to

\[
Bel(\tilde{x}_t) = \eta p(z_t \mid x_t) p(\tilde{x}_t \mid u_t, z_{t-1}, u_{t-1}, \ldots, u_0)
\]

- We can further expand the right-hand term based on the law of total probability

\[
Bel(\tilde{x}_t) = \eta p(z_t \mid x_t) \int p(\tilde{x}_t \mid x_{t-1}, u_t, \ldots, u_0) p(\tilde{x}_{t-1} \mid z_{t-1}, u_{t-1}, \ldots, u_0) dx_{t-1}
\]
Bayesian Filters (5)

• This is followed by again applying a Markov assumption to obtain

\[
Bel(\tilde{x}_i) = \eta p(z_i \mid x_i) \int p(\tilde{x}_i \mid u_i, x_{i-1}) p(\tilde{x}_{i-1} \mid z_{i-1}, u_{i-1}, \ldots, u_0) dx_{i-1}
\]

• But now the right hand term is merely our belief at the previous time step. Thus…

\[
Bel(\tilde{x}_i) = \eta p(z_i \mid x_i) \int p(\tilde{x}_i \mid u_i, x_{i-1}) Bel(\tilde{x}_{i-1}) dx_{i-1}
\]

… or our current belief is based upon integrating possible robot motions over our previous belief, and then conditioned to our current measurement.

Bayesian Filters (6)

\[
Bel(\tilde{x}_i) = \eta p(z_i \mid x_i) \int p(\tilde{x}_i \mid u_i, x_{i-1}) Bel(\tilde{x}_{i-1}) dx_{i-1}
\]

• This corresponds to both the prediction and correction phases of a Bayesian filter

• For the KF for example, the integration is accomplished in the predictive step where we add the covariance matrix \( Q \) associated with our process noise to the transformed covariance estimate \( P_{k-1} \) to obtain our prior “belief” \( P_k \)

• This is conditioned to the new sensor readings in the measurement update phase to obtain \( P_k \), our belief at time \( k \)
Monte Carlo Localization (Homework 5)

Predictor-Corrector Example

1) We have a prior of uniform weighted particle

\[ t = k^- \]

At this point, we have \( m \) unique samples

2) Particles are weighted based on the sensor measurement and resampled according to weight to generate our posterior.

\[ t = k \]

We still have \( m \) samples, but they are all equally weighted and not necessarily unique

3) Particles are passed through our motion model to generate a new posterior

\[ t = (k + 1)^- \]

Particles are again unique and equally weighted
The Particle Filter

- Unlike the KF, which represents the pdf parametrically as a gaussian, the PF approximates it as a sample set

\[ \text{Bel}(\bar{x}) \approx \{x^i, w^i\} \quad i \in [1..m] \]

- \(m\) denotes the number of particles in the sample set
- \(x^i\) corresponds to a hypothetical state estimate
- \(w^i\) corresponds to a weight reflecting a “confidence” in how well the particle \(x^i\) reflects the true state \(x\)

- \(\sum w^i = 1\), so that the sample set corresponds to a discrete probability density function

- It has been shown that as the number of samples approaches infinity, the sample set converges to the true posterior [Tanner, Tools for Statistical Inference, 1996]. However, no proofs for rates of convergence exist

Solving the Bayes Filter Equation

- Particle filter filters approximate the solution to our belief equation by a numerical integration over the particle set

\[ \text{Bel}(\bar{x}_i) = \eta p(z_i | x_i) \int p(\bar{x}_i | u_t, x_{t-1}) \text{Bel}(\bar{x}_{t-1}) dx_{t-1} \]

1. Choose a particle at random from the prior distribution
2. Project ahead by generating a new sample from the motion model
3. Repeat steps 1-2 \(m\) times
4. Reweight each sample based upon the new sensor measurement
5. Normalize the weight factors to sum to 1
Predictor-Corrector Example (2)

1) We have a prior of uniform weighted particle

\[ p_1 = k^- \]

![Diagram of robot positions](image)

Predictor-Corrector Example (3)

2a) Particles are weighted based on the sensor measurement

\[ w_1 = 0.4 \quad w_2 = 0.5 \quad w_3 = 0.1 \]

2b) Particles are resampled according to weight to generate our posterior.

\[ s_1 = 0.323 \quad s_2 = 0.677 \quad s_3 = 0.900 \]

\[ t = k \]

\[ p_1, p_2, p_3 \]

We still have \( m \) samples, but they are all equally weighted and not necessarily unique.
Predictor-Corrector Example (3)

3) Particles are passed through our motion model to generate a new posterior

\[ t = k + 1 \]

Particles are again unique and equally weighted

4) Iterate...

The Particle Filter Algorithm

\[
function \quad X_{k+1} = runFilter(X_k, z_k, u_k) \\
X_{k+1} = \emptyset \\
for \ i = 1 : m \\
\quad \text{generate random } x \text{ from } X \text{ based on sample weights; } \\
\quad \text{generate random } x' \sim p(x' \mid u_k, x); \\
\quad w = p(z_k \mid x'); \\
\quad \text{Insert } (x', w) \in X_{k+1}; \\
\quad \text{end } \\
\text{Normalize weight factors } \forall \ w_i \in X_{k+1}; \\
\text{return } X_{k+1};
\]
The Particle Filter Algorithm (ver. 2.0)

```plaintext
function X_{k+1} = runFilter (X_k, z_k, u_k)
X_{k+1} = \emptyset; \quad X_{\text{prop}} = \emptyset;
for i = 1 : m
    generate random \( x \) from \( X_k \) based on sample weights;
    Insert \( x \in X_{\text{prop}} \)
end
for all \( x \in X_{\text{prop}} \)
    \( x = x' \sim p(x' \mid u_k, x) \);
end
for all \( x \in X_{\text{prop}} \)
    \( w = p(z_k \mid x') \);
    Insert \((x, w) \in X_{k+1} \);
end
Normalize weight factors \( \forall w_i \in X_{k+1} \);
return \( X_{k+1} \);
```

This version will be better for your Matlab implementation on the PF assignment!

The MCL Problem

- In the MCL problem, our objective is to estimate position and orientation in a workspace
- We assume the availability of a map \( m \)
- This allows us to condition our belief to not only the current pose, but constrained to lie within a map. Thus, our belief equation becomes

\[
Bel(\tilde{x}_i) = p(z_i \mid x_i, m) \int p(\tilde{x}_i \mid u_i, x_{i-1}, m) Bel(\tilde{x}_{i-1}) d\tilde{x}_{i-1}
\]

- Thus, we can infer expected measurements from a given pose through:
  - Ray tracing if we are doing an occupancy grid
  - Line intersection if we are representing the map as a set of lines
- We can also combine map information with our motion model to exploit constraints in the workspace
Generating the Sensor Model

• Operation of the particle filter hinges upon associating a probability with each sensor measurement given a state so that a proper weight can be associated with each sample

$$\text{Bel}(\hat{x}_t) = \eta \int p(\hat{x}_t | u_t, x_{t-1}, m) \text{Bel}(\hat{x}_{t-1}) dx_{t-1}$$

• This is NOT the same as sampling the probability density function of $$z$$

• For a continuous distribution, the probability of measuring a specific value is zero

• Normally, sensors have a resolution which a given measurement is rounded to (e.g. a LRF may have a cm level resolution)

• Probabilities can then be determined by integrating the sensor pdf over this resolution range

Generating a MCL Specific Sensor Model

• MCL is typically performed with range sensors at known bearing angles to the robot (although cameras have also been used)

• As such, a single scan consists of numerous sensor measurements (e.g. from laser or sonar pulses)

• If we assume that these $$n$$ measurements are independent, the conditional probability can then be expressed as

$$p(z_i | x_i, m) = \prod_{i=1}^{n} p(z_i^i | x_i, m)$$
MCL Sensor Model Issues (1)

- A shortcoming of particle filters is that they tend to fail if the sensor models are too accurate
- This can result from not generating an initial sample close enough to the true state estimate
- One potential solution is to inflate the sensor model error. For example, the standard deviation for the SICK LRF is modeled as $\sigma=25\text{cm}$ when in reality it is closer $1\text{cm}$.
- This violates the basis from which the PF was derived, but has basis in actual measurements and works well in practice

![Graph of Particle Distribution](image)

MCL Sensor Model Issues (2)

- A second issue using the LRF is that for many scans, there will be no sensor data available
- This typically results from wall features being outside the maximum range of the sensor as above, but can also arise when the laser scan is absorbed, multi-path error, etc.
- To address this, the probability of obtaining such a reading is explicitly modeled. The weighting of this probability is a function of the range and the environment being explored

![Graph of Reading Probability](image)
Recall that the conditional probability for the sensor measurement is expressed as the product of the individual probabilities.

\[
p(z_t | x_t, m) = \prod_{i=1}^{n} p(z_i' | x_t, m)
\]

As a consequence, a single “outlier” can cause the probability to approach zero.

Such errors can readily be caused by errors in our map, furniture, persons/robots moving throughout the environment, etc.

This is handled by introducing an exponential based probability density into the sensor model for unmodeled “obstacles”.

---

**Sample Sensor Models**

- Ultrasound.
- Laser range-finder.
Simple MCL Examples

- MCL relies upon difference in the environment to induce corresponding differences in sensor measurements.
- Large open areas, long featureless corridors, symmetric environments, etc. can cause MCL to be slow to converge or to converge to the wrong pose.
- MCL can exploit even minor differences to obtain a correct pose estimate.

When can MCL Fail?

- MCL relies upon difference in the environment to induce corresponding differences in sensor measurements.
- Large open areas, long featureless corridors, symmetric environments, etc. can cause MCL to be slow to converge or to converge to the wrong pose.
- MCL can exploit even minor differences to obtain a correct pose estimate.

Inconsistent Convergence

Consistent Convergence
Generating a State Estimate

- Since each particle has an associated state and weight, the mean of the distribution can be estimated using standard techniques and serve as our estimate of the state.
- This suffers when there are competing distributions (when the filter has not yet converged).
- Alternately, one could use the particle with the highest weight.
- A drawback to this is that a single sample is used to generate the state estimate.
- A third alternative is to discretize the state space, find the cell with the highest total weight, and calculate the mean over this particle subset.
- Other methods can be imagined.

The Particle Filter Pros & Cons

- There are several advantages to using particle filters:
  - Able to model non-linear system dynamics and sensor models.
  - No gaussian noise model assumptions.
  - They can use implicit as well as parametric estimators.
  - In practice, performs well in the presence of large amounts of noise and assumption violations (e.g. Markov assumption, weighting model…).
  - Simple implementation.
- Some disadvantages include:
  - Higher computational complexity compared to the KF.
  - Computational complexity increases exponentially compared with increases in state dimension (typically NOT used outdoors).
  - In some applications, the filter is more likely to diverge with more accurate measurements.
Kalman Filter - Supporting References

- G. Welch & G. Bishop, “An Introduction to the Kalman Filter”
- P. Maybeck, “Stochastic Models, Estimation & Control,” Chapter 1 (attached to W&B tutorial)

Data Fusion Motivation (1)

- Let’s say your robot takes 3 range measurements of the distance to a beacon as $Z = [2000, 1900, 2100]^T$
- What would be your estimate of the beacon distance?
- Well, a good estimate might be the mean of the 3 sensor values:

$$ r = E(Z) = \frac{2000 + 1900 + 2100}{3} = 2000 $$
Data Fusion Motivation (2)

• Now let’s say your robot takes 3 measurements of the distance to a beacon as \( Z = [2000, 1900, 3100]^T \).
• We could again use the mean as the range estimate and obtain:

\[
r = E(Z) = \frac{\sum_{i=1}^{3} p_i z_i}{3} = \frac{2000 + 900 + 3100}{3} = 2000
\]

• Would you have as much confidence in this estimate as the first?

The Kalman Filter (1)

• The main idea behind the Kalman filter is that you do not just have an estimate for a parameter \( x \) but also have some estimate for the uncertainty in your value for \( x \).
• This is represented by the variance/covariance of the estimate \( P_x \).
• There are many advantages to this, as it allows you a means for estimating the confidence in your robot’s ability to execute a task (e.g. navigating through a tight doorway).
• In the case of the KF, it also provides a nice mechanism for optimally combining data over time.
• This optimality condition assumes we have linear models, and the error characteristics of our sensors can be modeled as zero-mean, Gaussian noise.
Notation Review

1. Matrices are denoted by a capital letter. In text, they will be bold (e.g. $A$).
2. Vectors are denoted by a lowercase letter. In text, they will be bold (e.g. $x$). In Microsoft Equations, they will have an overscore e.g. $\bar{x}_i$.
3. Scalars are lowercase letters without emphasis.
4. $x_k^a$ denotes the a priori estimate for the state vector $x$ at time step $k$ before the measurement update phase.
5. $x_k$ denotes the estimate for the state vector $x$ at time step $k$ after the measurement update phase.

The Discrete Kalman Filter

* The Kalman filter addresses the problem of estimating the state $x \in \mathbb{R}^n$ of a discrete-time controlled process governed by the linear difference equation

$$\tilde{x}_{k+1} = A\tilde{x}_k + Bu_k + \tilde{w}_k$$

and with a measurement $z \in \mathbb{R}^m$ that is

$$\tilde{z}_k = H\tilde{x}_k + \tilde{v}_k$$

* $w_k$ and $v_k$ represent the process and measurement noise. They are assumed independent, white, and with gaussian PDFs

$$p(w) \sim N(0, Q) \quad p(v) \sim N(0, R)$$

* $A, B, H, Q, R$ may be time varying.
The Predictor-Corrector Approach

- In this example, prediction comes from using knowledge of the vehicle dynamics to estimate its change in position.
- The analogy would be integrating information from the vehicle odometry or to estimate changed in position.
- The correction is accomplished through making exteroceptive observations and then fusing this with your current estimate.
- This is akin to updating position estimates using landmark information, etc.
- In practice, the prediction rate is typically much higher than the correction.

The Discrete Kalman Filter (2)

- At each time step, the KF propagates both a state estimate $x_k$ and an estimate for the error covariance $P_k$. The latter provides an indication of the uncertainty associated with the state estimate.
- As mentioned previously, the KF is a predictor-corrector algorithm. Prediction comes in the time update phase, and correction in the measurement update phase.

In our case, prediction will be from the robot kinematics ($vX, vY, \alpha$).
The Time Update Phase

1. Predict the state ahead

\[ \bar{x}_{k+1}^- = A\bar{x}_k + Bu_k \]

2. Project the error covariance ahead

\[ P_{k+1}^- = AP_k A^T + Q \]

The Measurement Update Phase

1. Compute the Kalman Gain \( K_k \)

2. Update the estimate based on the new measurement \( z_k \)

\[ \bar{x}_k = \bar{x}_k^- + K (z_k - H\bar{x}_k^-) \]

3. Update the error covariance

\[ P_k = (I - K_k H)P_k^- \]
Predictor-Corrector KF Example (1)

1) We have a \textit{covariance} matrix $P$ with mean $x_k$. $x_k$ is our pose estimate and the $P$ is the uncertainty associated with that pose estimate.

2) We predict the next position from our motion model

$$\begin{align*}
x_{k+1}^- &= A x_k + B u_k \\
P_{k+1}^- &= A P_k A^T + Q
\end{align*}$$

Predictor-Corrector KF Example (2)

3) We take a new measurement in the MU phase…

… and use this to estimate our new position $x_k$ and covariance $P_{k+1}$

$$\begin{align*}
K_k &= P_{k+1}^+ H (H P_{k+1}^+ H^T + R)^{-1} \\
\hat{x}_k &= \hat{x}_{k+1}^- + K_k (z_k - H \hat{x}_{k+1}^-) \\
P_k &= (I - K_k H) P_{k+1}^-
\end{align*}$$
The Kalman Filter (2)

- Step 1 in the time update phase is merely our prediction based upon the linear state update equation that we have:

\[ \hat{x}_{k+1}^- = Ax_k + Bu_k \]

- Step 2 of the time update phase comes from projecting our covariance matrix forward where we merely add the process noise variance \( Q \) due to the normal sum distribution property where \( \sigma_v^2 = \sigma_{\text{noise}}^2 + \sigma_{\text{process}}^2 \):

\[ P_{k+1} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^T (x_i - \mu) \]

\[ P_{k+1} = \frac{1}{N} \sum_{i=1}^{N} (x_{i|k} - \mu)(x_{i|k} - \mu)^T + Q \]

\[ P_{k+1} = \frac{1}{N} \sum_{i=1}^{N} [A(x_i - \mu)] [A(x_i - \mu)]^T + Q \]

\[ P_{k+1} = \frac{1}{N} \sum_{i=1}^{N} A(x_i - \mu)(x_i - \mu)^T A^T + Q \]

\[ P_{k+1} = AP_k A^T + Q \]

The Kalman Filter (3)

- Prediction before Measurement Update

- Measurement Update

\[ \text{Estimate after Measurement Update} \]

© JR Spletzer
The Correction: A Least-Squares Approach

- OK, let’s say we have two independent sensors and obtain two different measurements 
  \[ Z = [z_1, z_2]^T \] for the range \( r \) to a beacon
- Let us further assume that the variance in each of these sensor measurements is \( R_1 \) and \( R_2 \), respectively
- Q: How should we fuse these measurements in order to obtain the “best” possible resulting estimate for \( r \)?
- We’ll define “best” from a least-squares perspective…
- We have 2 measurements that are equal to \( r \) plus some additive zero-mean Gaussian noise \( v_1 \) and \( v_2 \)

\[
\begin{align*}
  z_1 &= r + N(0, R_1) = r + v_1 \\
  z_2 &= r + N(0, R_2) = r + v_2
\end{align*}
\]

A Least-Squares Approach

- We want to fuse these measurements to obtain a new estimate for the range \( \hat{r} \)
- Using a weighted least-squares approach, the resulting sum of squares error will be

\[
e = \sum_{i=1}^{n} w_i (\hat{r} - z_i)^2
\]

- Minimizing this error with respect to \( \hat{r} \) yields

\[
\frac{\partial e}{\partial \hat{r}} = \frac{\partial}{\partial \hat{r}} \sum_{i=1}^{n} w_i (\hat{r} - z_i)^2 = 2 \sum_{i=1}^{n} w_i (\hat{r} - z_i) = 0
\]
A Least-Squares Approach

- Rearranging we have

\[ \sum_{i=1}^{n} w_i \hat{r}_i - \sum_{i=1}^{n} w_i z_i = 0 \]

\[ \Rightarrow \hat{r} = \frac{\sum_{i=1}^{n} w_i z_i}{\sum_{i=1}^{n} w_i} \]

- If we choose the weight to be

\[ w_i = \frac{1}{\sigma_i^2} = \frac{1}{R_i} \]

we obtain

\[ \hat{r} = \frac{z_1 + z_2}{R_1 + 1 + \frac{R_2}{R_1}} = \frac{R_1}{R_1 + R_2} z_1 + \frac{R_2}{R_1 + R_2} z_2 \]

This can be rewritten as

\[ \hat{r} = z_1 + \frac{R_i}{R_i + R_2} (z_2 - z_1) \]

Kalman Gain

or if we think of this as adding a new measurement to our current estimate of the state we would get

\[ \hat{x}_{k+1} = \hat{x}_k + P_{k+1} \left( z_{k+1} - \hat{r}_{k+1} \right) \]

\[ \Rightarrow \hat{x}_{k+1} = \hat{x}_k + K_{k+1} (z_{k+1} - \hat{r}_{k+1}) \]

- For merging Gaussian distributions, the update rule is

\[ \frac{1}{\sigma_1^2} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} = \frac{\sigma_1^2 + \sigma_2^2}{\sigma_1^2 \sigma_2^2} \Rightarrow \sigma_1^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2} \]

which if we write in our measurement update equation form we get

\[ P_{k+1} = \frac{P_{k+1} R_{k+1}}{P_{k+1} R_{k+1} + R_{k+1}} \Rightarrow P_{k+1} - K_{k+1} P_{k+1} \]

Measurement Update

1. Compute Kalman Gain

\[ K_k = P_k H^T (HP^T R + H)^{-1} \]

2. Update state estimate with measurement \( z_k \)

\[ \hat{x}_k = \hat{x}_k + K_k (z_k - H \hat{x}_k) \]

3. Update error covariance

\[ P_k = (I - K_k H) P_k \]
1-D Example
Estimating a Random Constant

- Suppose we are trying to estimate the value of a 1D constant from corrupted sensor measurements. Our process model is then

\[ x_{k+1} = Ax_k + Bu_k + w_k = x_k + w_k \]

\[ z_k = Hx_k + v_k = x_k + v_k \]

- The KF equations then are

**Time Update**

\[ \hat{x}_{k+1} = x_k \]

\[ P_{k+1}^{-} = P_k - Q \]

**Measurement Update**

\[ K_k = P_k (P_k + R)^{-1} \]

\[ x_k = \hat{x}_k + K (z_k - \hat{x}_k) \]

\[ P_k = (I - K_k) P_k^- \]

- Let us assume that \( x^* = 7.5, Q = 0.01, R = 9 \)

- With perfect knowledge of the process and sensor covariance model, we obtain

![Simulation Results](image)
Let us assume that \( x^* = 7.5, \ Q = 0.01, \ R = 9 \)

Let us further assume that the user believes that the sensor covariance \( R = 900 \)

Simulation Results (3)

Let us assume that \( x^* = 7.5, \ Q = 0.01, \ R = 9 \)

Let us further assume that the user believes that the sensor covariance \( R = 900 \)
Kalman Filters vs. Particle Filters

<table>
<thead>
<tr>
<th>KF</th>
<th>PF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compact representation</td>
<td>Memory-intensive representation</td>
</tr>
<tr>
<td>Single state hypothesis</td>
<td>$n$ hypotheses (1 for each particle)</td>
</tr>
<tr>
<td>Explicitly model Gaussian PDF for state / covariance estimation</td>
<td>Implicitly Approximates any PDF for state/covariance estimation</td>
</tr>
<tr>
<td>Scales well computationally for higher dimensional representations</td>
<td>Limited to ~3 dimensions on modern computers</td>
</tr>
<tr>
<td>Diverge in the kidnapped robot problem</td>
<td>Solves the kidnapped robot problem</td>
</tr>
<tr>
<td>Limited to linear system models</td>
<td>Works for any system model</td>
</tr>
<tr>
<td>Optimal</td>
<td>Sub-optimal</td>
</tr>
</tbody>
</table>

Kalman Filter Localization (1)

- Let’s say that we are going to use a Kalman filter to localize our robot from range measurement to RF beacons with known location.

- We could write our state update equation as

\[
\begin{bmatrix}
    x_{k+1} \\
    y_{k+1} \\
    \theta_{k+1}
\end{bmatrix} =
\begin{bmatrix}
    x_k \\
    y_k \\
    \theta_k
\end{bmatrix} +
\begin{bmatrix}
    v \cos \theta \Delta t \\
    v \sin \theta \Delta t \\
    \omega \Delta t
\end{bmatrix}
\]

- Now let’s look at our measurement equations taking range to a beacon at $(x_b, y_b)$

\[
r_k = \sqrt{(x_k - x_b)^2 + (y_k - y_b)^2}
\]

- Houston, we have a problem…
• For many applications, the time update and measurement equations are NOT linear. As a consequence, the KF is not applicable.

• However, the KF is such a nice algorithm that maybe if we linearize around the non-linearities, we can still get good performance in practice.

• This line of thought lead to the development of the Extended Kalman Filter (EKF).

• By relaxing the linear assumptions, the use of the KF is extended dramatically.

• Life Rule: There is no such thing as a free lunch.

• We can no longer use the word “optimal” with the EKF.

---

The Extended Kalman Filter (EKF)

• The Extended Kalman (EKF) is a sub-optimal extension of the original KF algorithm.

• The EKF allows for estimation of non-linear processes or measurement relationships.

• This is accomplished by linearizing the current mean and covariance estimates (similar to a first order Taylor series approximation).

• Suppose our process and measurement equations are the non-linear functions:

\[
\begin{align*}
\bar{x}_{k+1} &= A\bar{x}_k + Bu_k + \bar{w}_k \\
\bar{z}_k &= H\bar{x}_k + \bar{v}_k \\
\bar{x}_{k+1} &= f(\bar{x}_k, u_k, \bar{w}_k) \\
\bar{z}_k &= h(\bar{x}_k, \bar{v}_k)
\end{align*}
\]

Kalman Filter  Extended Kalman Filter
For the state update equation, we do not know the noise values at each time step. So, we approximate the state and without them:

\[ \hat{x}_{k+1} = f(\hat{x}_k, \hat{u}_k, 0) \]

**EKF Time Update Phase (1)**

- **Time Update**
  1. Project the state forward: \( \hat{x}_{k+1}^- = f(\hat{x}_k, \hat{u}_k, 0) \)
  2. Project the covariance forward

- **Measurement Update**
  1. Compute Kalman Gain
  2. Update state estimate with measurement \( z_k \)
  3. Update error covariance

**EKF Time Update Phase (2)**

- However when we propagate the covariance ahead in time, the underlying function needs to be **linear** in order to properly combine the **Gaussian** uncertainty in our state \( x \) – our covariance matrix \( P_{k+1} \) with our process uncertainty \( Q \)
- Q: How do you think we could do this?
  - A: Linearization.

Again, our wonderful friend the Taylor series comes to the rescue 😊

\[ f(x_{k+1}) = f(x_k + \epsilon) = f(x_k) + f'(x_k)\epsilon + \ldots \]

\[ f(\hat{x}_{k+1}) = f(\hat{x}_k + \epsilon) = f(\hat{x}_k) + J\epsilon + \ldots \]
### Transforming Uncertainty (1)

- Let’s say we know the uncertainty of a variable $x$, and we want to compute the uncertainty of $y = f(x)$
- We know that $\bar{x} = \mu + \epsilon$
  where $\mu$ is the distribution mean and $\epsilon$ is zero mean noise
- We can then use the Jacobian to linearly approximate $y$
  \[ \hat{y} = f(x) = f(\bar{x} + \epsilon) \approx f(\bar{x}) + J\epsilon \]
- The mean of the distribution would then be
  \[ \bar{y} = E[\hat{y}] = E[f(\bar{x}) + J\epsilon] = f(\bar{x}) \]
- Therefore
  \[ \bar{y} - \bar{y} \approx J\epsilon \]

### Transforming Uncertainty (2)

- The covariance of the transformed distribution would then be
  \[ C_y = E[(\hat{y} - \bar{y})(\hat{y} - \bar{y})^T] = E[J\epsilon\epsilon^T J^T] = JC_x J^T \]
- Thus, to transform uncertainty across a non-linear transformation, we perform a similarity transform with the Jacobian
- Note that because of the $\approx$ symbols on the previous page, normal distributions are NOT preserved
- The optimality/robustness of the KF allows the EKF to work well in practice
EKF Time Update Phase (3)

- So the covariance is projected ahead as
  \[ P_{k+1}^- = AP_kA^T + Q \]
  
  where \( A \) is now the Jacobian of \( f \) with respect to \( x \) and \( W \) is the Jacobian of \( f \) with respect to \( w \).

EKF Robot Implementation Example (1)

- Assume that we have a mobile robot using odometry and range measurements to landmark to estimate its position and orientation
  \[ \bar{x} = [x, y, \theta]^T \]

- Assume that the odometry provides a velocity estimate \( V \) and an angular velocity estimate \( \omega \) that are both corrupted by gaussian noise

- We can write the state update equation as
  \[
  \begin{bmatrix}
  x_{k+1} \\
  y_{k+1} \\
  \theta_{k+1}
  \end{bmatrix} =
  \begin{bmatrix}
  x_k \\
  y_k \\
  \theta_k
  \end{bmatrix} +
  \begin{bmatrix}
  V \cos \theta \Delta t \\
  V \sin \theta \Delta t \\
  \omega \Delta t
  \end{bmatrix}
  \]

  which is obviously non-linear in the state.
EKF Robot Implementation Example (2)

• We calculate the Jacobian $A$ as

$$f(\vec{x}) = \begin{bmatrix} f_1(\vec{x}) \\ f_2(\vec{x}) \\ f_3(\vec{x}) \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} V \cos \theta \Delta t \\ V \sin \theta \Delta t \end{bmatrix} \Delta t$$

$$A = \frac{\partial f}{\partial x_j} = \begin{bmatrix} 1 & 0 & -V \sin \theta \Delta t \\ 0 & 1 & V \cos \theta \Delta t \\ 0 & 0 & 1 \end{bmatrix}$$

• We calculate the Jacobian $W$ from the sensor measurements as

$$W = \frac{\partial f}{\partial \vec{v}} = \begin{bmatrix} \Delta t \cos \theta & 0 \\ \Delta t \sin \theta & 0 \\ 0 & \Delta t \end{bmatrix}$$

EKF Measurement Update Phase (1)

• Again, in the measurement update we can have a non-linear relationship between our measurements and state

$$\vec{z}_k = h(\vec{x}_k, \vec{v}_k)$$

and once again we will assume that the noise is zero

$$\vec{z}_k = h(\vec{x}_k, 0)$$

• To propagate uncertainty, we shall again have to calculate the appropriate Jacobians

- $H$ is the Jacobian relating changes in $h$ to changes in our state $x$
- $V$ is the Jacobian relating changes in $h$ to changes in the measurement noise $v$

• These are then substituted into the original KF as appropriate
EKF Measurement Update Phase (2)

**KF**

* Computing the Kalman Gain:
  \[ K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \]

**EKF**

* Computing the Kalman Gain:
  \[ K_k = P_k^- H^T (H P_k^- H^T + VRV^T)^{-1} \]

* State Update:
  \[ \hat{x}_k = \hat{x}_k^- + K_k (\bar{z}_k - H\hat{x}_k^-) \]

* Covariance Update:
  \[ P_k = (I - K_k H) P_k^- \]

NOTE: Some derivations will write this as \( Hx \) as well.

Kalman Filter Localization Example Revisited

* Let’s go back to our beacon example and let’s assume we have a single beacon. From this, our range measurement can be written as
  \[ z_k = h(\bar{x}) = \sqrt{(x_k - x_b)^2 + (y_k - y_b)^2} \]

  where \( x_b \) and \( y_b \) are constants

* The Jacobians \( H \) and \( V \) can then be calculated as
  \[ H = \frac{\partial z}{\partial x} = \frac{x_k - x_b}{\sqrt{(x_k - x_b)^2 + (y_k - y_b)^2}} \quad \frac{y_k - y_b}{\sqrt{(x_k - x_b)^2 + (y_k - y_b)^2}} \]

  \[ V = 1 \]

* Note that if this were the only measurements available then \( \theta \) would be unobservable

* Q: How do we address this?
The Discrete Extended Kalman Filter

- **Time Update**
  1. Project the state forward
     \[ \hat{x}_k^\prime = f(\hat{x}_{k-1}, u_k, 0) \]
  2. Project the covariance forward
     \[ P_k^\prime = A_k P_{k-1} A_k^T + W_k Q_k W_k^T \]

- **Measurement Update**
  1. Compute Kalman Gain
     \[ K_k = P_k^\prime H_k^T (H_k P_k^\prime H_k^T + R_k V_k V_k^T)^{-1} \]
  2. Update state estimate with measurement \( z_k \)
     \[ \hat{x}_k = \hat{x}_k^\prime + K_k (z_k - h(\hat{x}_k^\prime, 0)) \]
  3. Update error covariance
     \[ P_k = (I - K_k H_k) P_k^\prime \]

Issues with the EKF

- The EKF approximates the KF using linearization about the current state estimate
- These linearizations can be particularly problematic in estimating robot orientation
- Hybrid approaches often use KF based solutions to estimate position and a particle filter based for estimating orientation (even viable in 3D)
- Linearizations also tend to yield an overconfident estimate of the covariance in the state estimate – meaning that the robot thinks its pose estimate is better than it actually is
  - Can address this to a point by artificially inflating the \( Q \) & \( R \) matrices
- In practice, the covariance matrices have to be “tuned” empirically to optimize system performance
5.8.1 Simultaneous Localization and Mapping (EPFL)

- Small local error accumulate to arbitrary large global errors!
- This is usually irrelevant for navigation
- However, when closing loops, global error does matter

5.8.2 Cyclic Environments

Courtesy of Sebastian Thrun
Summary (1)

- For localization indoors, particle filter approaches dominate
  - Unlike EKF approaches, they do not need a good initial estimate for localization
  - They are also able to solve the kidnapped robot problem by continuously distributing a small number of additional particles during each iteration
- For localization off the plane, the EKF is the preferred approach
  - Computationally they scale cubicly in the dimension of the problem vs. exponentially for the PF
- Both particle filters and EKF are used to attack the simultaneous localization and mapping (SLAM) problem
  - Again PFs can work well indoors for this
  - EKF can also attack this problem since the origin of the map can be assigned to the initial robot pose.

Summary (2)

- For map building, EKFs rely on tracking specific features in the environment.
  - This requires solving the correspondence problem (matching) in prior to running the EKF
  - In practice, EKFs can track 1000-2000 features in real-time (remember $O(n^3)$ is still not a nice complexity bound)
  - This constrains the size and fidelity of the maps that can be generated
  - The EKF still remains the algorithm of choice for SLAM outdoors as you won’t obtain nice line scans of walls to match for your PF
- Two primary issues with mapping:
  - Cycles
  - Dynamic environments