ME 599/699 Robot Modeling & Control
State Estimation and SLAM

Hasan Poonawala

State Estimation

Challenge

How do we extract $x(t)$ from our measurements?

There are a few options:

1. Bayesian Inference [Eg. Kalman Filter]
2. Implicitly invert the forward map $y(t) = h(x(t))$ using dynamics and history [Eg. Luenberger Observer]
3. Explicitly invert the forward map to get $x(t) = h^{-1}(y(t))$ [Eg. Encoder-Decoder archs. in ML, Computer Vision]

Bayesian Methods

We consider time to be discrete, corresponding to specific motion and measurement events.

Bayesian Methods

We consider time to be discrete, corresponding to specific motion and measurement events.

The state x over some set of time instants $0, 1, 2, \ldots, t$ is represented by $\mathbf{x}_{0:t}$

Bayesian Methods

We consider time to be discrete, corresponding to specific motion and measurement events.

The state x over some set of time instants $0, 1, 2, \ldots, t$ is represented by $\mathbf{x}_{0:t}$

We don't know the true values $\mathbf{x}_{0:t}$, which includes current state x_t and past states.

Bayesian Methods

We consider time to be discrete, corresponding to specific motion and measurement events.

The state x over some set of time instants $0, 1, 2, \ldots, t$ is represented by $\mathbf{x}_{0:t}$

We don't know the true values $\mathbf{x}_{0:t}$, which includes current state x_t and past states.

We represent this lack of certain knowledge by treating x_t at each time t as a **random variable**.

Uncertain State

Instead of saying our state x has a specific value, say

$$x = \begin{bmatrix} 1 \\ 3.4 \end{bmatrix},$$

we say that the state is a random variable $\mathbf{X}$ with continuous/discrete probability distribution $p_{\mathbf{X}}(x)$.

Uncertain State

Instead of saying our state x has a specific value, say

$$x = \begin{bmatrix} 1 \\ 3.4 \end{bmatrix},$$

we say that the state is a random variable $\mathbf{X}$ with continuous/discrete probability distribution $p_{\mathbf{X}}(x)$.

If $\mathbf{X}$ is discrete, then the probability that our random variable $\mathbf{X}$ has value x is $p_{\mathbf{X}}(x) = \Pr(X = x)$

Uncertain State

Instead of saying our state x has a specific value, say

$$x = \begin{bmatrix} 1 \\ 3.4 \end{bmatrix},$$

we say that the state is a random variable $\mathbf{X}$ with continuous/discrete probability distribution $p_{\mathbf{X}}(x)$.

If $\mathbf{X}$ is discrete, then the probability that our random variable $\mathbf{X}$ has value x is $p_{\mathbf{X}}(x) = \Pr(X = x)$

For example, let $\mathbf{X}$ be a random variable that can have integer values.
Then, we can speak of the probability that $\mathbf{X} = 4$, or $\mathbf{X} = 19283$, denoted as $p_{\mathbf{X}}(4)$ and $p_{\mathbf{X}}(19283)$ respectively.

Example: Coin Toss

A coin toss has two outcomes: heads (H) or tails (T).

The result of tossing the coin C is unknown in advance, and we therefore **model** it as a random variable with possible values $\{H, T\}$

Example: Coin Toss

A coin toss has two outcomes: heads (H) or tails (T).

The result of tossing the coin C is unknown in advance, and we therefore **model** it as a random variable with possible values $\{H, T\}$

We describe this random variable C using the probability distribution function

$$p_C = \begin{cases} p & \text{if } C = H \\ 1 - p & \text{if } C = T \end{cases}$$

A fair coin has $p = 0.5$, otherwise the coin is biased.

Example: Coin Toss

A coin toss has two outcomes: heads (H) or tails (T).

The result of tossing the coin C is unknown in advance, and we therefore **model** it as a random variable with possible values $\{H, T\}$

We describe this random variable C using the probability distribution function

$$p_C = \begin{cases} p & \text{if } C = H \\ 1 - p & \text{if } C = T \end{cases}$$

A fair coin has $p = 0.5$, otherwise the coin is biased.

Similarly, a dice D has six outcomes $\{1, 2, 3, 4, 5, 6\}$, and we use five numbers to describe the uncertainty in single rolls of a dice.

Uncertain State

When $\mathbf{x}$ is continuous, like a real-valued vector, we describe the uncertainty in its value using a probability **density** function.

Uncertain State

When $\mathbf{x}$ is continuous, like a real-valued vector, we describe the uncertainty in its value using a probability **density** function.

A common situation is when $\mathbf{x}$ is a real number.

Uncertain State

When **x** is continuous, like a real-valued vector, we describe the uncertainty in its value using a probability **density** function.

A common situation is when **x** is a real number.

We may ask: what is the probability that the random variable **X** has a specific value $x \in \mathbb{R}$?
Usually, $p_{\mathbf{X}}(x) = \Pr(X = x) = 0$.

Uncertain State

When **x** is continuous, like a real-valued vector, we describe the uncertainty in its value using a probability **density** function.

A common situation is when **x** is a real number.

We may ask: what is the probability that the random variable **X** has a specific value $x \in \mathbb{R}$?
Usually, $p_{\mathbf{X}}(x) = \Pr(X = x) = 0$.

Instead of x having a specific value, we ask whether x **belongs to a set**.

Uncertain State

When **x** is continuous, like a real-valued vector, we describe the uncertainty in its value using a probability **density** function.

A common situation is when **x** is a real number.

We may ask: what is the probability that the random variable **X** has a specific value $x \in \mathbb{R}$?
Usually, $p_{\mathbf{X}}(x) = \Pr(X = x) = 0$.

Instead of x having a specific value, we ask whether x **belongs to a set**.

Consider the real interval (a, b)
Then,

$$\Pr(X \in (a, b)) = \int_a^b p_{\mathbf{X}}(x) dX$$

Gaussian Random Variable

A common probability density function is the Gaussian distribution:

$$p_{\mathbf{x}}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

- μ is the average value of the distribution
- σ is the standard deviation.
- σ^2 is called the variance.
- Notation: $x \sim \mathcal{N}(\mu, \sigma^2)$

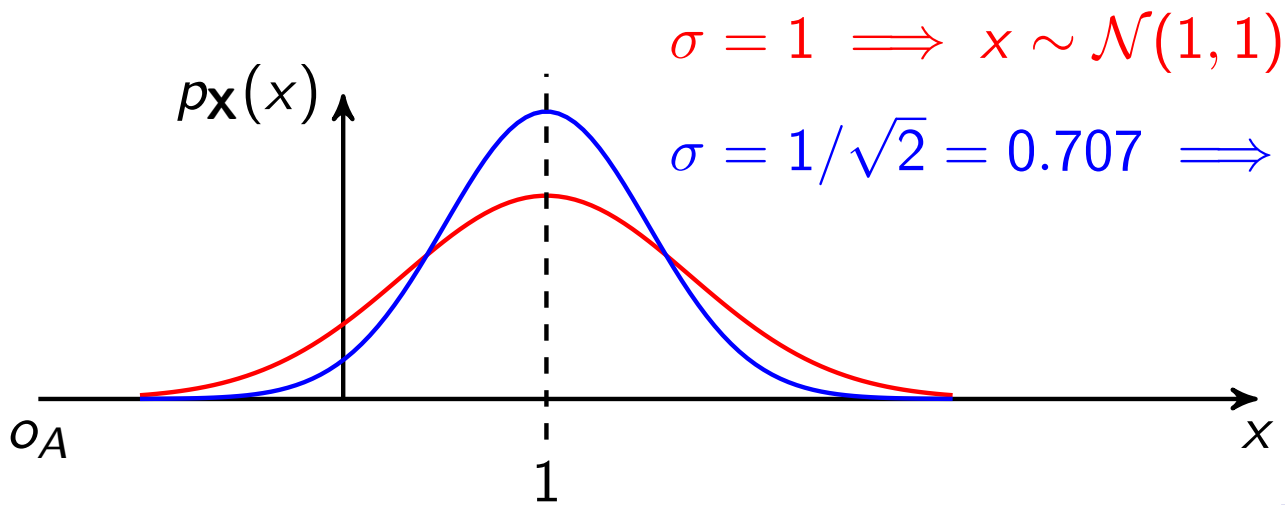
Gaussian Random Variable

A common probability density function is the Gaussian distribution:

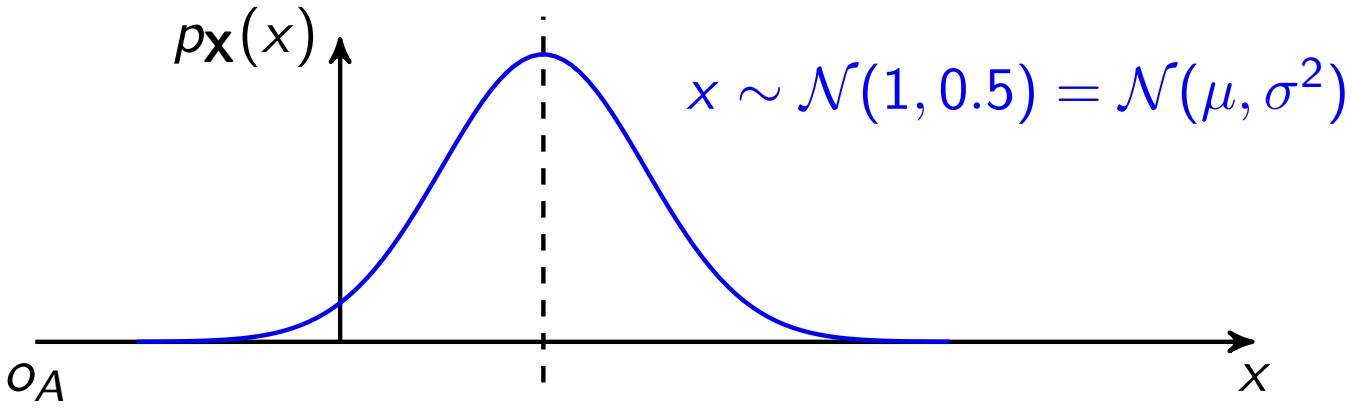
$$p_{\mathbf{X}}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

- μ is the average value of the distribution
- σ is the standard deviation.
- σ^2 is called the variance.
- Notation: $x \sim \mathcal{N}(\mu, \sigma^2)$

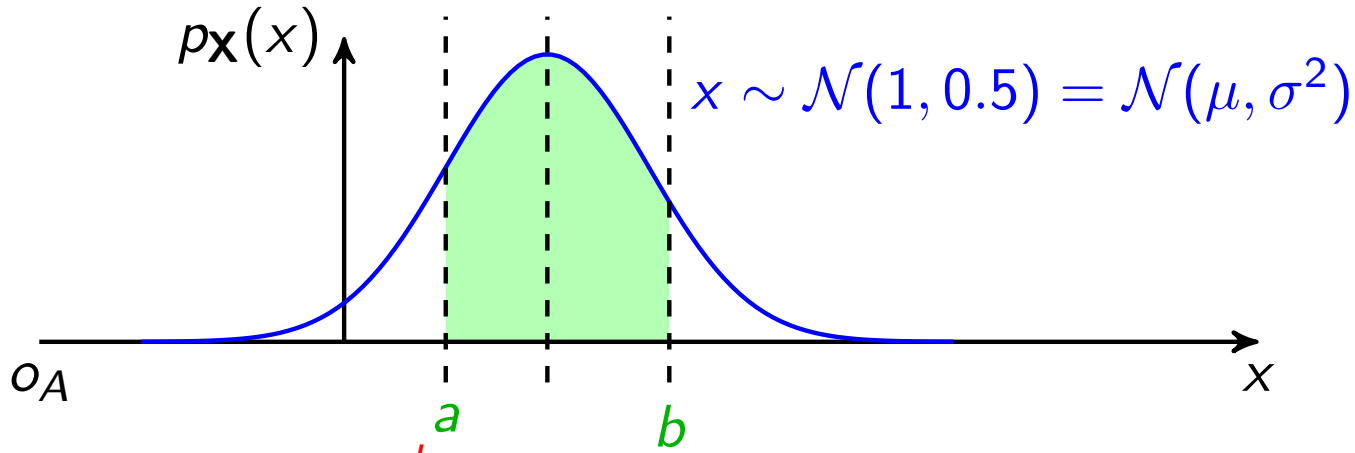
We can plot this function $p_{\mathbf{X}}(x)$:



Uncertainty regions

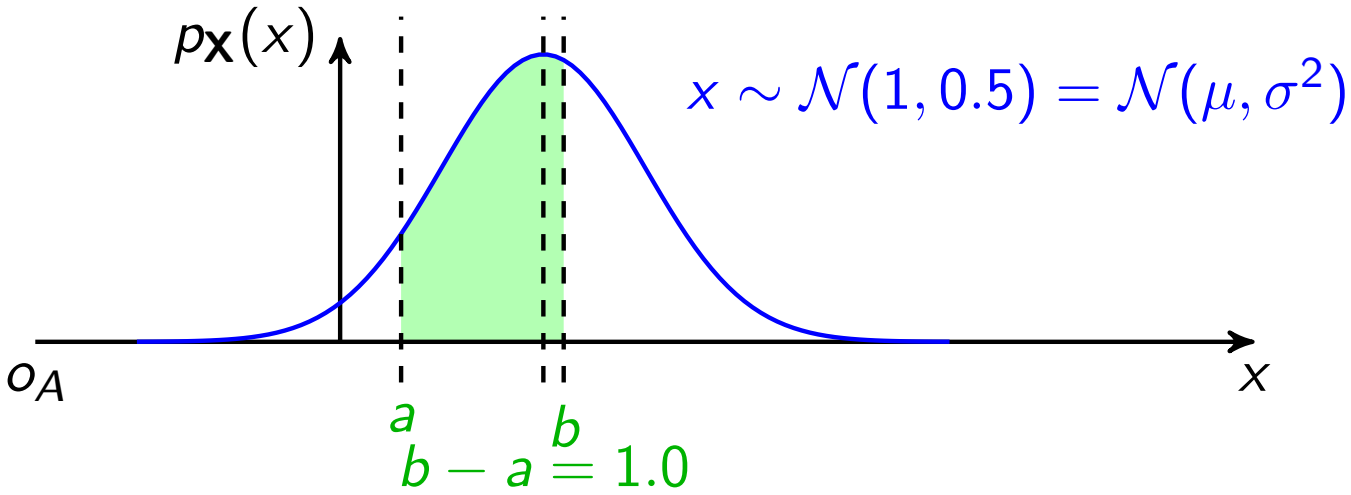


Uncertainty regions

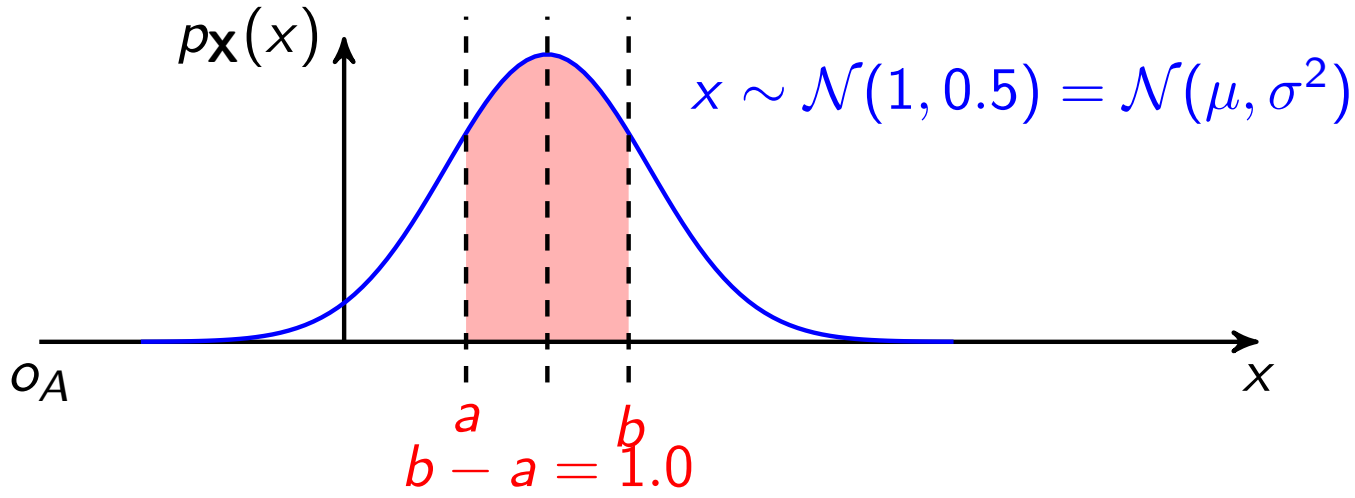


$\Pr(x \in (a, b)) = \int_a^b p_{\mathbf{X}}(x)dx =$ area under curve

Uncertainty regions

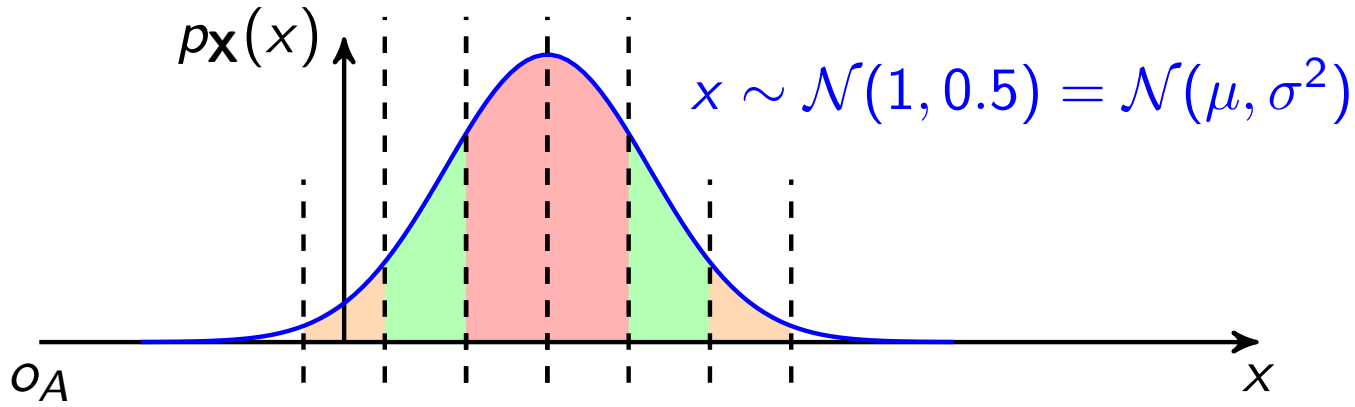


Uncertainty regions



Same interval, larger probability when centered at μ

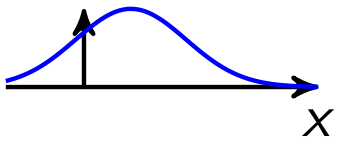
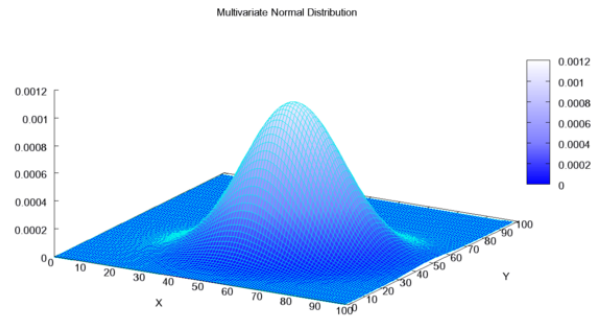
Uncertainty Regions



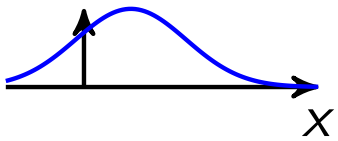
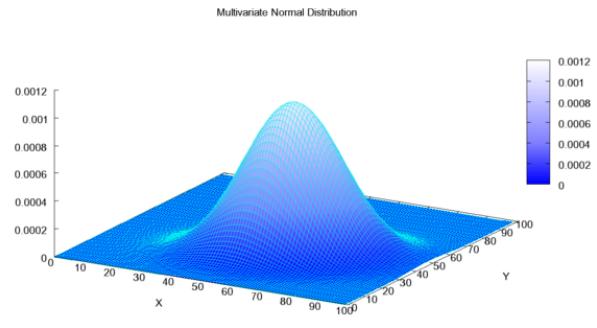
Consider the interval $I(k) = (\mu - k\sigma, \mu + k\sigma)$:
length $2k\sigma$ centered at μ.

- $\Pr(x \in I(1)) = 0.682$ (red)
- $\Pr(x \in I(2)) = 0.954$ (red + green)
- $\Pr(x \in I(3)) = 0.997$ (red + green + orange)

Multivariate Gaussians

	$\mathbb{R}$-valued Gaussian	$\mathbb{R}^n$-valued Gaussian
$p_{\mathbf{X}}(x)$	$\frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$ 	$\frac{1}{(2\pi \det(\Sigma))^{\frac{n}{2}}} \exp^{-\frac{1}{2}\left((x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$ 
$I(k)$	$\left(\frac{x-\mu}{\sigma}\right)^2 \leq k^2$ (interval in $\mathbb{R}$)	$(x-\mu)^T\Sigma^{-1}(x-\mu) \leq k^2$ (ellipse in $\mathbb{R}^n$)

Multivariate Gaussians

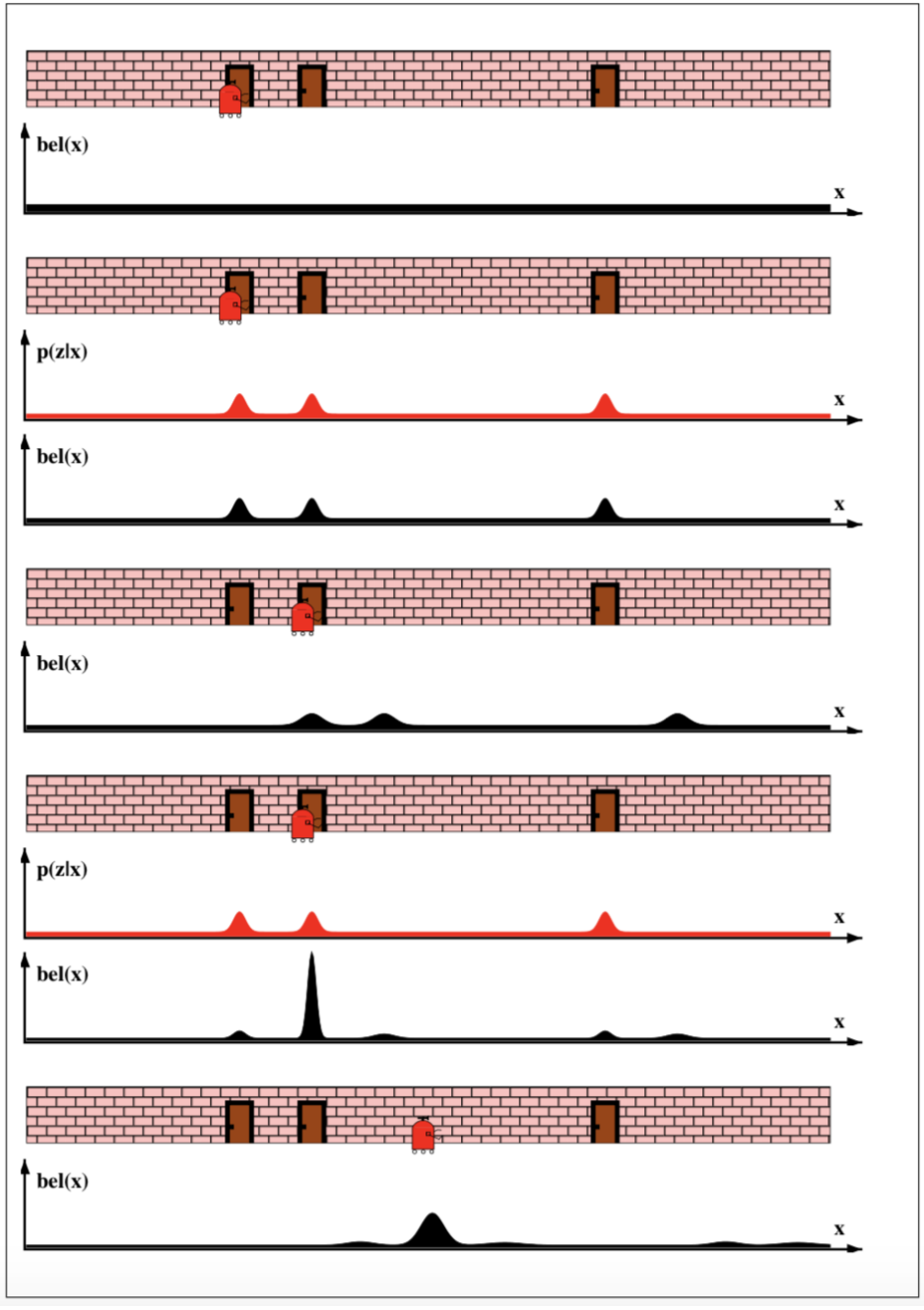
	$\mathbb{R}$-valued Gaussian	$\mathbb{R}^n$-valued Gaussian
$p_{\mathbf{X}}(x)$	$\frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$ 	$\frac{1}{(2\pi \det(\Sigma))^{\frac{n}{2}}} \exp^{-\frac{1}{2}\left((x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$ 
$I(k)$	$\left(\frac{x-\mu}{\sigma}\right)^2 \leq k^2$ (interval in $\mathbb{R}$)	$(x-\mu)^T\Sigma^{-1}(x-\mu) \leq k^2$ (ellipse in $\mathbb{R}^n$)

Main takeaway: We represent $p_{\mathbf{X}}(x) \sim \mathcal{N}(\mu, \Sigma)$ as an ellipse in X, instead of $p_{\mathbf{X}}(x)$ vs x.

Center of ellipse $\leftrightarrow$ mean μ

Shape/Size of ellipse $\leftrightarrow$ Covariance Σ

Bayesian Methods



Initial uniform belief on x

Observe a door. We model probability of observing door as $p(z|x)$

Update $p(x)$; matches $p(z|x)$

Update $p(x)$ due to motion, peaks 'flatten'

Observe a door again.

Bayes update makes being at second door highly likely.

Motion update flattens peaks

Bayesian Methods

Main idea: A model of what we expect to measure in a state tells us how to infer where we are, using measurements.

Bayesian Methods

Main idea: A model of what we expect to measure in a state tells us how to infer where we are, using measurements.

Bayes' Rule:

$$p(\text{state}|\text{measurement}) = \frac{p(\text{measurement}|\text{state})p(\text{state})}{p(\text{measurement})}$$

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

Bayesian Methods

Main idea: A model of what we expect to measure in a state tells us how to infer where we are, using measurements.

Bayes' Rule:

$$p(\text{state}|\text{measurement}) = \frac{p(\text{measurement}|\text{state})p(\text{state})}{p(\text{measurement})}$$

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

In words: If we express our uncertainty about the state x as $p(x)$,

Bayesian Methods

Main idea: A model of what we expect to measure in a state tells us how to infer where we are, using measurements.

Bayes' Rule:

$$p(\text{state}|\text{measurement}) = \frac{p(\text{measurement}|\text{state})p(\text{state})}{p(\text{measurement})}$$

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

In words: If we express our uncertainty about the state x as $p(x)$, and we have a model $p(y|x)$ of how measurement y depends on state x,

Bayesian Methods

Main idea: A model of what we expect to measure in a state tells us how to infer where we are, using measurements.

Bayes' Rule:

$$p(\text{state}|\text{measurement}) = \frac{p(\text{measurement}|\text{state})p(\text{state})}{p(\text{measurement})}$$

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

In words: If we express our uncertainty about the state x as $p(x)$, and we have a model $p(y|x)$ of how measurement y depends on state x, then we can update our model of uncertain state using a measurement.

Bayesian Methods

Main idea: A model of what we expect to measure in a state tells us how to infer where we are, using measurements.

Bayes' Rule:

$$p(\text{state}|\text{measurement}) = \frac{p(\text{measurement}|\text{state})p(\text{state})}{p(\text{measurement})}$$

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

In words: If we express our uncertainty about the state x as $p(x)$, and we have a model $p(y|x)$ of how measurement y depends on state x, then we can update our model of uncertain state using a measurement.

The updated uncertainty is p(state 'given' measurement), or $p(x|y)$, or p(state 'conditioned on' measurement)

Motion Model

Secondary idea: When the robot moves, $p(x_t)$ and $p(x_{t+1})$ should depend on this motion, even if we never measure anything.

Motion Model

Secondary idea: When the robot moves, $p(x_t)$ and $p(x_{t+1})$ should depend on this motion, even if we never measure anything.

Given an input u_t at time t, we need a **motion model** $p(x_{t+1}|x_t, u_t)$

In summary, we have two types of updates to our model of the uncertain state:

1. Update due to motion
2. Update due to measurement

Kalman Filter

The Kalman Filter was developed for discrete-time linear time-invariant dynamical systems driven by noise:

$$x_{t+1} = Ax_t + Bu_t + v_t; \quad y_t = Cx_t + w_t,$$

where v_t and w_t represent zero-mean gaussian noise with covariances V and W respectively.

Kalman Filter

The Kalman Filter was developed for discrete-time linear time-invariant dynamical systems driven by noise:

$$x_{t+1} = Ax_t + Bu_t + v_t; \quad y_t = Cx_t + w_t,$$

where v_t and w_t represent zero-mean gaussian noise with covariances V and W respectively.

- Motion model: $x_{t+1} = Ax_t + Bu_t + v_t$
- Sensor model: $y_t = Cx_t + w_t$

Kalman Filter

The Kalman Filter was developed for discrete-time linear time-invariant dynamical systems driven by noise:

$$x_{t+1} = Ax_t + Bu_t + v_t; \quad y_t = Cx_t + w_t,$$

where v_t and w_t represent zero-mean gaussian noise with covariances V and W respectively.

- Motion model: $x_{t+1} = Ax_t + Bu_t + v_t$
- Sensor model: $y_t = Cx_t + w_t$

$p(x_0)$ is assumed to be a Gaussian random variable [only one peak!]: we have a mean μ_0 and covariance matrix Σ_0

Kalman Filter

The Kalman Filter was developed for discrete-time linear time-invariant dynamical systems driven by noise:

$$x_{t+1} = Ax_t + Bu_t + v_t; \quad y_t = Cx_t + w_t,$$

where v_t and w_t represent zero-mean gaussian noise with covariances V and W respectively.

- Motion model: $x_{t+1} = Ax_t + Bu_t + v_t$
- Sensor model: $y_t = Cx_t + w_t$

$p(x_0)$ is assumed to be a Gaussian random variable [only one peak!]: we have a mean μ_0 and covariance matrix Σ_0

At time t, $p(x_t|\mathbf{y}_{0:t}, \mathbf{u}_{0:t-1})$ will also be Gaussian, and our job is to come up with an appropriate mean μ_t and variance Σ_t using inputs and measurements.

Kalman Filter

The Kalman Filter was developed for discrete-time linear time-invariant dynamical systems driven by noise:

$$x_{t+1} = Ax_t + Bu_t + v_t; \quad y_t = Cx_t + w_t,$$

where v_t and w_t represent zero-mean gaussian noise with covariances V and W respectively.

- Motion model: $x_{t+1} = Ax_t + Bu_t + v_t$
- Sensor model: $y_t = Cx_t + w_t$

$p(x_0)$ is assumed to be a Gaussian random variable [only one peak!]: we have a mean μ_0 and covariance matrix Σ_0

At time t, $p(x_t|\mathbf{y}_{0:t}, \mathbf{u}_{0:t-1})$ will also be Gaussian, and our job is to come up with an appropriate mean μ_t and variance Σ_t using inputs and measurements.

The Kalman filter equations tell us the best way to combine the uncertain motion and uncertain measurement, given these models and noise properties.

Kalman Filter Algorithm

System :

$$x_{t+1} = Ax_t + Bu_t + v_t \tag{1}$$

$$y_t = Cx_t + w_t \tag{2}$$

$$\hat{x}_t \sim \mathcal{N}(\mu_t, \Sigma_t), w_t \sim \mathcal{N}(0, R_w), v_t \sim \mathcal{N}(0, R_v) \tag{3}$$

1. Motion update

$$\mu_t^{pred} = A\mu_t + Bu_t \tag{4}$$

$$\Sigma_t^{pred} = A\Sigma_t A^T + R_v \tag{5}$$

2. Measurement update

$$K_t = \Sigma_t^{pred} C^T \left(C\Sigma_t^{pred} C^T + R_w \right)^{-1} \tag{6}$$

$$\mu_{t+1} = \mu_t^{pred} + K(y_t - C\mu_t^{pred}) \tag{7}$$

$$\Sigma_{t+1} = (I - KC)\Sigma_t^{pred} \tag{8}$$

Example: 1D Point Mass KF

System $\ddot{m}q(t) = u$ has discrete-time state

$$x_t = \begin{bmatrix} q_t \\ v_t \end{bmatrix}$$

where v_t is the velocity of the mass.

The discrete-time dynamics, with discretization time T is

$$x_{t+1} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ \frac{T}{m} \end{bmatrix} u_t + v_t \tag{9}$$

$$y_t = Cx_t + w_t \tag{10}$$

If we measure position, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$.
If we measure velocity, $C = \begin{bmatrix} 0 & 1 \end{bmatrix}$.
If we measure both, $C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ or $C = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Example: 1D Point Mass KF

Performance (Run `Julia` code on Canvas):

- Measuring position using very noisy sensors won't work
- Measuring position using decent sensors works well enough
- Measuring velocity with noisy sensors is terrible
- Measuring velocity with decent sensors is better, but still has the issue of (slow or fast) drift in position over time. This drift is undetected by the filter.
- Conclusion: we need some form of measurement of position

Simultaneous Localization and Mapping

A robot is typically unable to measure its position.

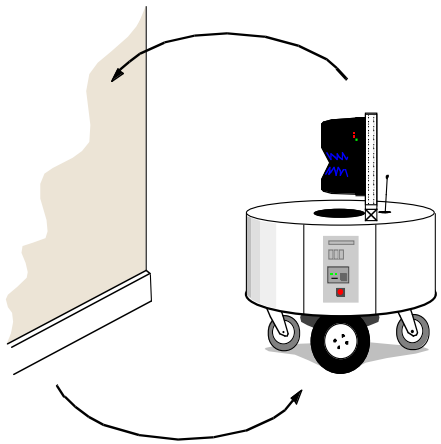
Instead, it can measure its position in relation to objects in the world.

Can't use KF directly to estimate location when we don't have a map of objects in the the world.

Without a map, we don't know what to expect as measurement in a state (no sensor model).

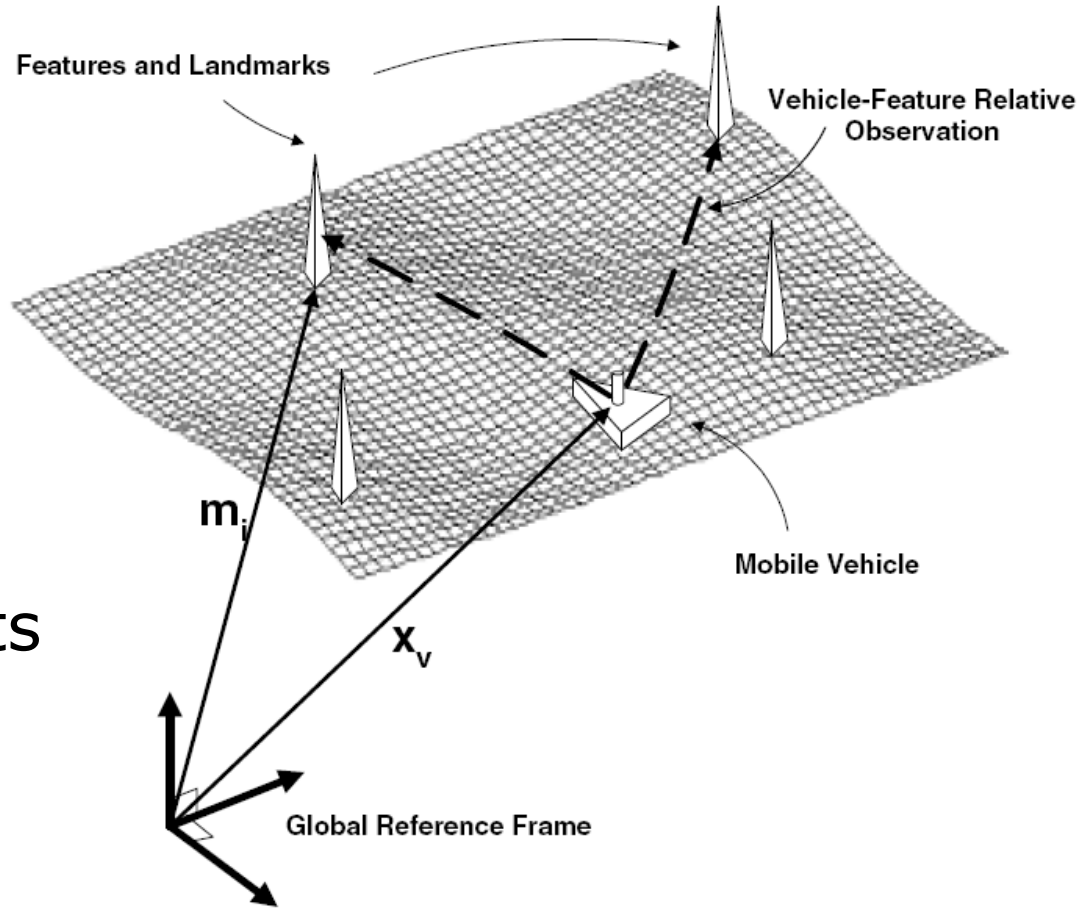
The SLAM Problem

- SLAM is a **chicken-or-egg** problem:
  - → a map is needed for localization and
  - → a pose estimate is needed for mapping



Feature-Based SLAM

- **Absolute** robot poses
- **Absolute** landmark positions
- But only **relative** measurements of landmarks



Feature-Based SLAM

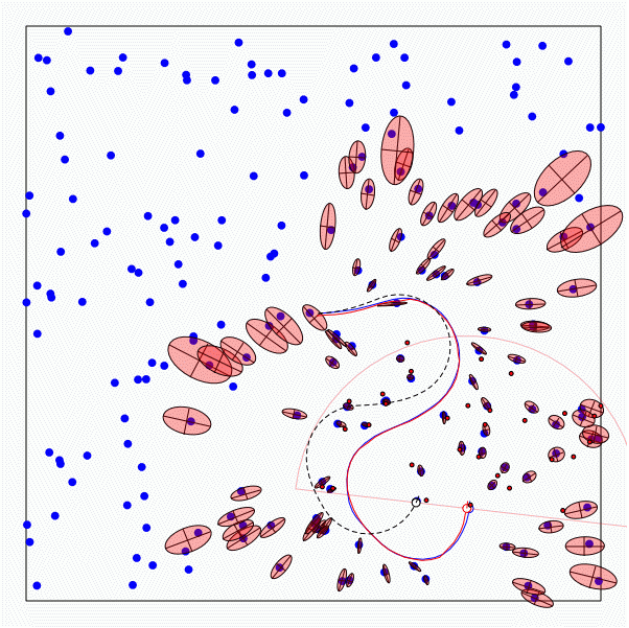
Given:

- The robot's controls

$$U_{1:k} = \{u_1, u_2, \ldots, u_k\}$$

- Relative observations

$$Z_{1:k} = \{z_1, z_2, \ldots, z_k\}$$



Wanted:

- Map of features

$$m = \{m_1, m_2, \ldots, m_n\}$$

- Path of the robot

$$X_{1:k} = \{x_1, x_2, \ldots, x_k\}$$

Simultaneous Localization and Mapping

SLAM tries to simultaneously make sense of where we think we are (localization) and what we think we should be seeing (mapping).

Basic idea: Make the map a part of the state.

We can build measurement and motion models for this expanded state.

Example: 2D SLAM Non-rotating robot

The robot at (x_r, y_r) measures the location of landmarks $l_i = (l_{ix}, l_{iy})$ in its frame of reference, which is aligned with the world axis.

State is $x = \begin{bmatrix} x_r \\ y_r \\ l_{1x} \\ l_{1y} \\ l_{2x} \\ l_{2y} \\ \vdots \\ l_{n_l x} \\ l_{n_l y} \end{bmatrix}$ where there are n_l landmarks.

Example: 2D SLAM Non-rotating robot

The landmarks are assumed stationary, with robot moving according to

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix}_{t+1} = \begin{bmatrix} x_r \\ y_r \end{bmatrix}_t + u_t + v_t$$

Therefore, A is $(2 + 2n_l) \times (2 + 2n_l)$, the only non-zero elements being

$$A_{1,1} = 1, \quad A_{2,2} = 1$$

.

B is $(2 + 2n_l) \times 2$, the only non-zero elements being

$$B_{1,1} = 1, \quad B_{2,2} = 1$$

.

Example: 2D SLAM Non-rotating robot

If the robot sees all landmarks in its frame, its measurement is

$$y = \begin{bmatrix} l_{1x} - x_r \\ l_{1y} - y_r \\ l_{2x} - x_r \\ l_{2y} - y_r \\ \vdots \end{bmatrix}$$

Therefore, C is $2n_l \times (2 + 2n_l)$:

$$C = \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & & & & & \vdots & & \\ -1 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}$$

Example: 2D SLAM Non-rotating robot

Performance when robot always sees all landmarks (Run `Julia` code on Canvas):

- The estimated mean position of each landmark is quite close to the actual landmark position, relative to other landmarks
- The uncertainty in the landmark position is identical to the uncertainty in initial position, in the long run
  - The robot can treat the landmark as lying within the 3σ ellipse of the landmark mean. To avoid collision with high probability, the 3σ ellipse corresponding to its position should lie outside the landmark's ellipse.
- The difference between the robot's true and estimated location depends on sensor accuracy

Extended Kalman Filter

For linear systems with gaussian additive noise, the Kalman filter provides the least uncertain estimate.
When the system is nonlinear:

$$x_{t+1} = f(x_t, u_t) + v_t \tag{11}$$
$$y_t = h(x_t) + w_t, \tag{12}$$

use linearization:

$$A_t = \left.\frac{\partial f}{\partial x}\right|_{x=\mu_t}, \quad B_t = \left.\frac{\partial f}{\partial u}\right|_{u=u_t}, \quad C_t = \left.\frac{\partial h}{\partial x}\right|_{x=\mu_t^{pred}}.$$

A blind application of Kalman filter updates using A_t, B_t, and C_t is not guaranteed to work but often does well in practice.

Example: 2D SLAM Rotating robot

The inputs are the speed s of the robot along its current heading direction, and the angular velocity ω.

$$u_t = \begin{bmatrix} s_t \\ \omega_t \end{bmatrix}$$

The motion model is now nonlinear:

$$\begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix}_{t+1} = \begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix}_t + \begin{bmatrix} s_t \cos\theta_r \\ s_t \sin\theta_r \\ \omega_t \end{bmatrix} + v_t = f(x_t, u_t)$$

We linearize to get part of the A and B matrices

$$\begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix}_{t+1} = \begin{bmatrix} 1 & 0 & -\sin\theta_r s_t \\ 0 & 1 & -\cos\theta_r s_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix}_t + \begin{bmatrix} \cos\theta_r & 0 \\ \sin\theta_r & 0 \\ 0 & 1 \end{bmatrix} u_t + v_t$$

Example: 2D SLAM Rotating robot

The position of a landmark l_i in the robot's frame becomes

$$y_i = \begin{bmatrix} \cos\theta_r & -\sin\theta_r \\ \sin\theta_r & \cos\theta_r \end{bmatrix} \begin{bmatrix} l_{ix} - x_r \\ l_{iy} - y_r \end{bmatrix}.$$

The full measurement y corresponds to stacking the y_is.

The measurement model is nonlinear: $y = h(x) \neq Cx$ for any matrix C.

As mentioned earlier, we linearize

$$C = \left.\frac{\partial h}{\partial x}\right|_{x=\mu_t^{pred}}$$

Limited Sensing

We've assumed that we can see all landmarks at once.

In practice, we can see nearby landmarks.

This limitation creates a data association problem when landmarks look similar to each other. For example, the entry to an office cubicle.

EKF SLAM: Building the Map

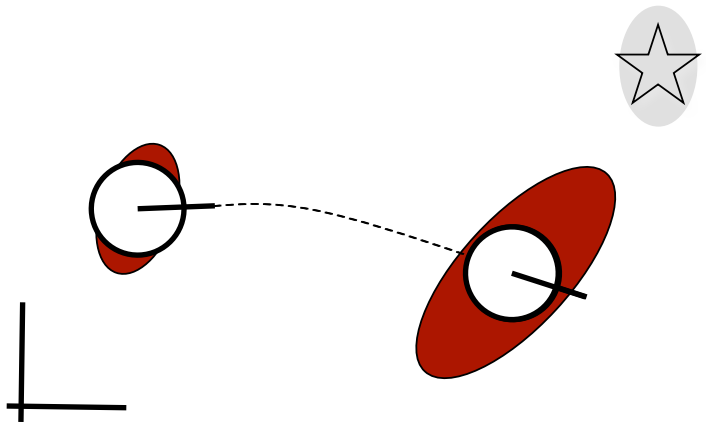
Filter Cycle, Overview:

1. State prediction (odometry)
2. Measurement prediction
3. Observation
4. Data Association 

5. Update
6. Integration of new landmarks

EKF SLAM: Building the Map

- State Prediction



Odometry:

$$\hat{\mathbf{x}}_R = f(\mathbf{x}_R, \mathbf{u})$$

$$\hat{C}_R = F_x C_R F_x^T + F_u U F_u^T$$

Robot-landmark cross-covariance prediction:

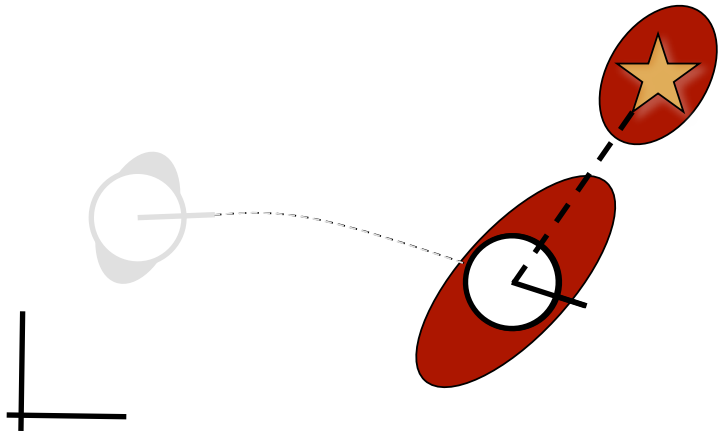
$$\hat{C}_{RM_i} = F_x C_{RM_i}$$

(skipping time index k)

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}_k \qquad C_k = \begin{bmatrix} C_R & C_{RM_1} & C_{RM_2} & \cdots & C_{RM_n} \\ C_{M_1R} & C_{M_1} & C_{M_1M_2} & \cdots & C_{M_1M_n} \\ C_{M_2R} & C_{M_2M_1} & C_{M_2} & \cdots & C_{M_2M_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{M_nR} & C_{M_nM_1} & C_{M_nM_2} & \cdots & C_{M_n} \end{bmatrix}_k$$

EKF SLAM: Building the Map

- Measurement Prediction



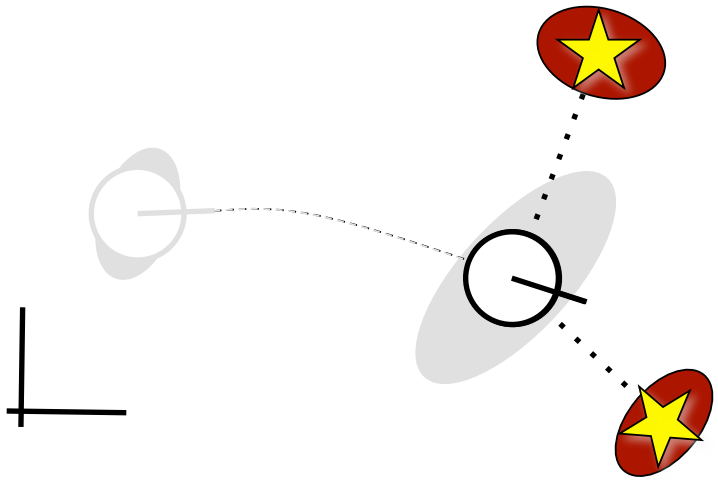
Global-to-local frame transform h

$$\hat{\mathbf{z}}_k = h(\hat{\mathbf{x}}_k)$$

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}_k \qquad C_k = \begin{bmatrix} C_R & C_{RM_1} & C_{RM_2} & \cdots & C_{RM_n} \\ C_{M_1R} & C_{M_1} & C_{M_1M_2} & \cdots & C_{M_1M_n} \\ C_{M_2R} & C_{M_2M_1} & C_{M_2} & \cdots & C_{M_2M_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{M_nR} & C_{M_nM_1} & C_{M_nM_2} & \cdots & C_{M_n} \end{bmatrix}_k$$

EKF SLAM: Building the Map

- Observation



(x,y)-point landmarks

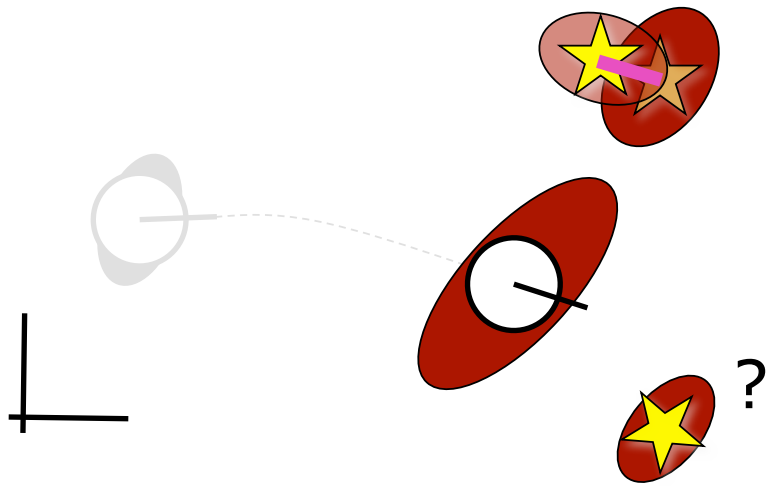
$$\mathbf{z}_k = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix}$$

$$R_k = \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix}$$

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}_k \qquad C_k = \begin{bmatrix} C_R & C_{RM_1} & C_{RM_2} & \cdots & C_{RM_n} \\ C_{M_1R} & C_{M_1} & C_{M_1M_2} & \cdots & C_{M_1M_n} \\ C_{M_2R} & C_{M_2M_1} & C_{M_2} & \cdots & C_{M_2M_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{M_nR} & C_{M_nM_1} & C_{M_nM_2} & \cdots & C_{M_n} \end{bmatrix}_k$$

EKF SLAM: Building the Map

- Data Association



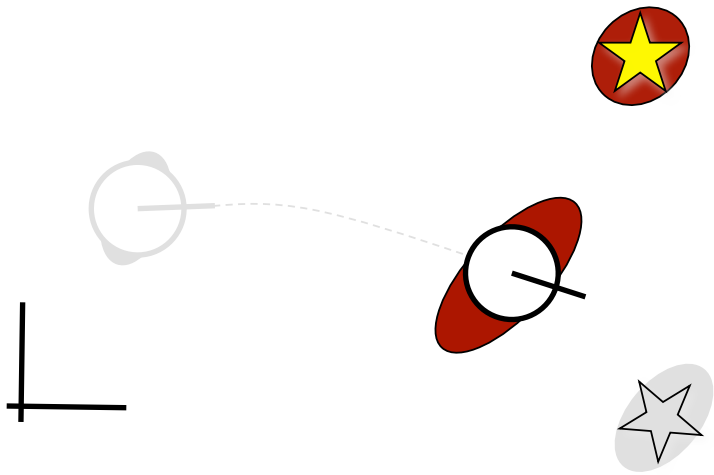
Associates predicted measurements $\hat{\mathbf{z}}_k^i$ with observation $\mathbf{z}_k^j$

$$\begin{aligned} \nu_k^{ij} &= \mathbf{z}_k^j - \hat{\mathbf{z}}_k^i \\ S_k^{ij} &= R_k^j + H^i \hat{C}_k H^{iT} \end{aligned}$$

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}_k \qquad C_k = \begin{bmatrix} C_R & C_{RM_1} & C_{RM_2} & \cdots & C_{RM_n} \\ C_{M_1R} & C_{M_1} & C_{M_1M_2} & \cdots & C_{M_1M_n} \\ C_{M_2R} & C_{M_2M_1} & C_{M_2} & \cdots & C_{M_2M_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{M_nR} & C_{M_nM_1} & C_{M_nM_2} & \cdots & C_{M_n} \end{bmatrix}_k$$

EKF SLAM: Building the Map

- Filter Update



The usual Kalman filter expressions

$$K_k = \hat{C}_k \, H^T S_k^{-1}$$

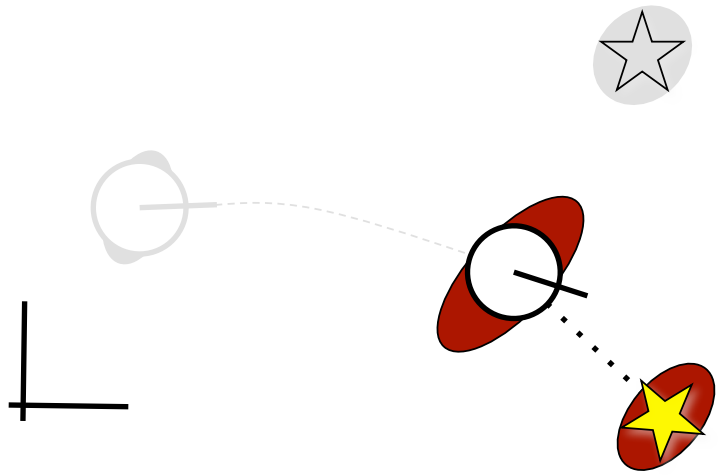
$$\mathbf{x}_k = \hat{\mathbf{x}}_k + K_k \, \nu_k$$

$$C_k = (I - K_k \, H) \, \hat{C}_k$$

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}_k \qquad C_k = \begin{bmatrix} C_R & C_{RM_1} & C_{RM_2} & \cdots & C_{RM_n} \\ C_{M_1R} & C_{M_1} & C_{M_1M_2} & \cdots & C_{M_1M_n} \\ C_{M_2R} & C_{M_2M_1} & C_{M_2} & \cdots & C_{M_2M_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{M_nR} & C_{M_nM_1} & C_{M_nM_2} & \cdots & C_{M_n} \end{bmatrix}_k$$

EKF SLAM: Building the Map

- Integrating New Landmarks



State augmented by

$$\mathbf{m}_{n+1} = g(\mathbf{x}_R, \mathbf{z}_j)$$

$$C_{M_{n+1}} = G_R\, C_R\, G_R^T + G_z\, R_j\, G_z^T$$

Cross-covariances:

$$C_{M_{n+1}M_i} = G_R\, C_{RM_i}$$

$$C_{M_{n+1}R} = G_R\, C_R$$

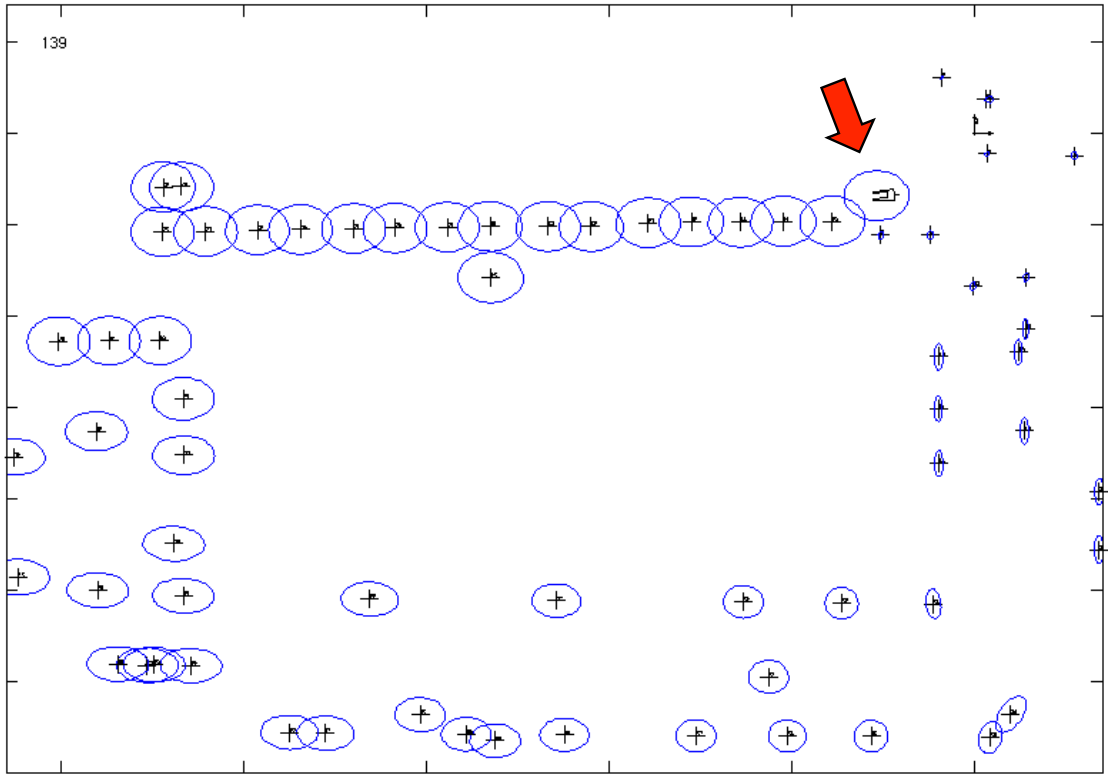
$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \\ \mathbf{m}_{n+1} \end{bmatrix}_k \qquad C_k = \begin{bmatrix} C_R & C_{RM_1} & C_{RM_2} & \cdots & C_{RM_n} & C_{RM_{n+1}} \\ C_{M_1R} & C_{M_1} & C_{M_1M_2} & \cdots & C_{M_1M_n} & C_{M_1M_{n+1}} \\ C_{M_2R} & C_{M_2M_1} & C_{M_2} & \cdots & C_{M_2M_n} & C_{M_2M_{n+1}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{M_nR} & C_{M_nM_1} & C_{M_nM_2} & \cdots & C_{M_n} & C_{M_nM_{n+1}} \\ C_{M_{n+1}R} & C_{M_{n+1}M_1} & C_{M_{n+1}M_2} & \cdots & C_{M_{n+1}M_n} & C_{M_{n+1}} \end{bmatrix}_k$$

SLAM: Loop Closure

- **Recognizing an already mapped area**, typically after a long exploration path (the robot "closes a loop”)
- Structurally identical to data association, but
  - high levels of ambiguity
  - possibly useless validation gates
  - environment symmetries
- Uncertainties **collapse** after a loop closure (whether the closure was correct or not)

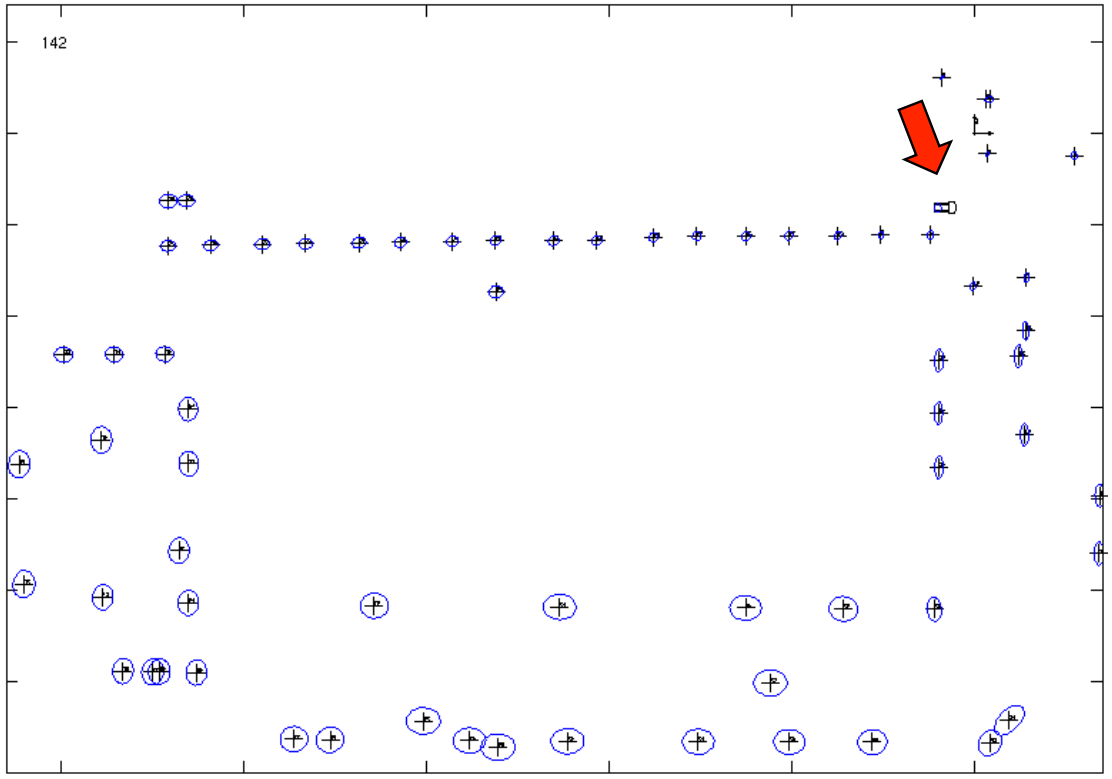
SLAM: Loop Closure

- Before loop closure



SLAM: Loop Closure

- After loop closure



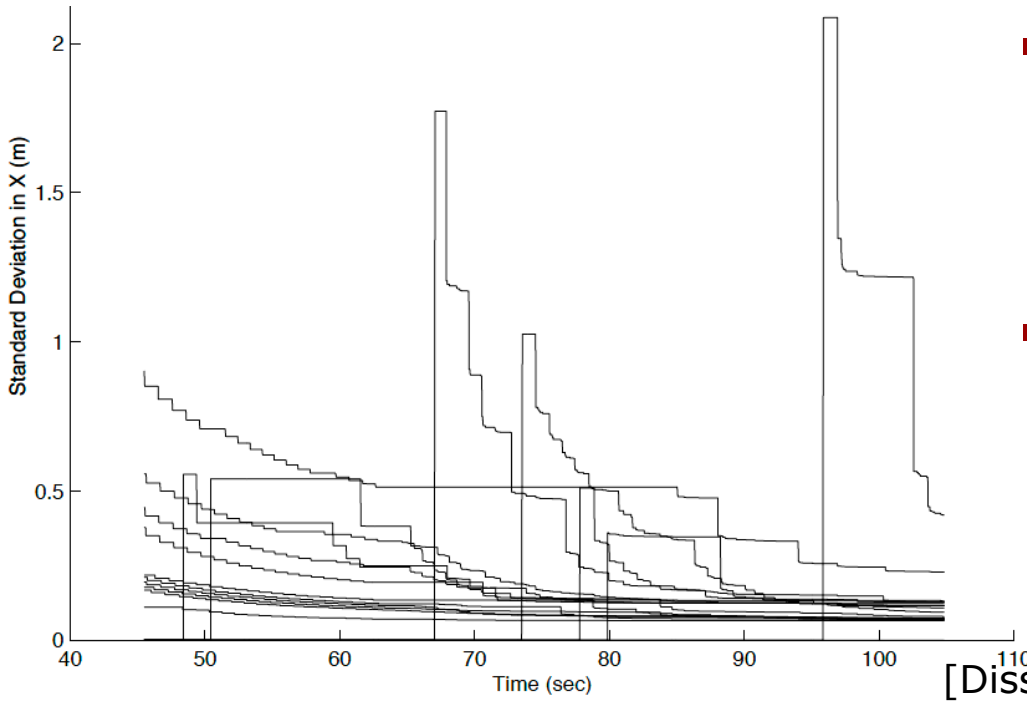
SLAM: Loop Closure

- By revisiting already mapped areas, uncertainties in robot and landmark estimates can be **reduced**
- This can be exploited when **exploring** an environment for the sake of better (e.g. more accurate) maps
- Exploration: the problem of ***where* to acquire new information**

→ See separate chapter on exploration

KF-SLAM Properties (Linear Case)

- The **determinant** of any sub-matrix of the map covariance matrix **decreases monotonically** as successive observations are made

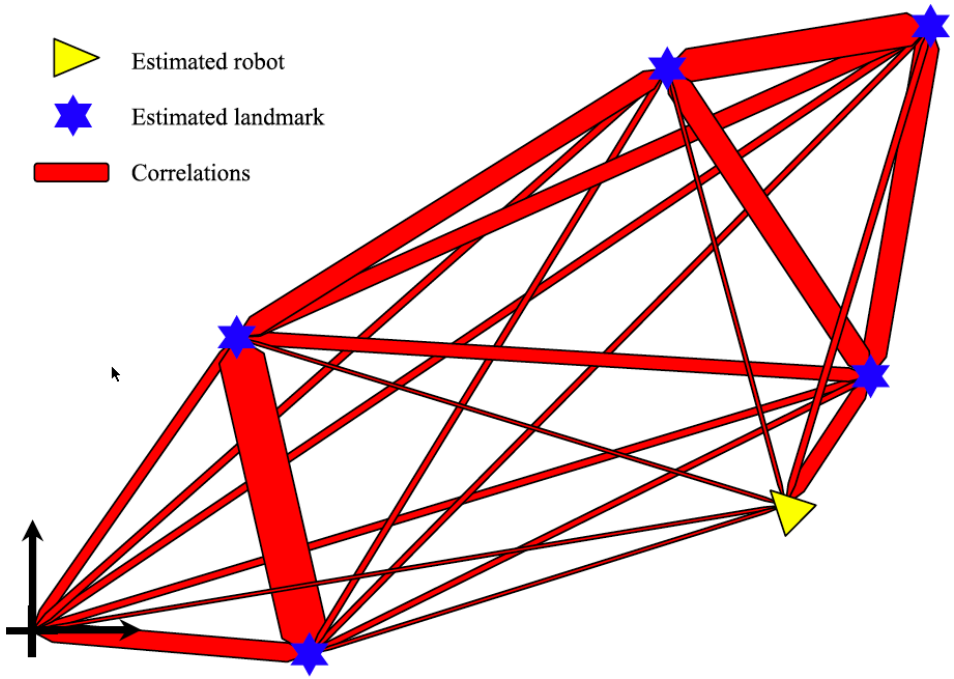


- When a new landmark is initialized, its **uncertainty** is **maximal**
- Landmark uncertainty **decreases monotonically** with each new observation

[Dissanayake et al., 2001]

KF-SLAM Properties (Linear Case)

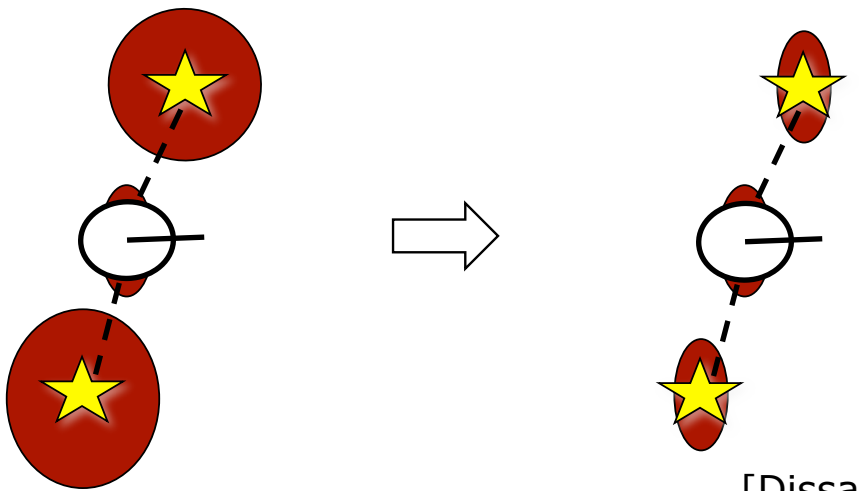
- In the limit, the landmark estimates become **fully correlated**



[Dissanayake et al., 2001]

KF-SLAM Properties (Linear Case)

- In the limit, the **covariance** associated with any single landmark location estimate is determined only by the **initial covariance in the vehicle location estimate**.



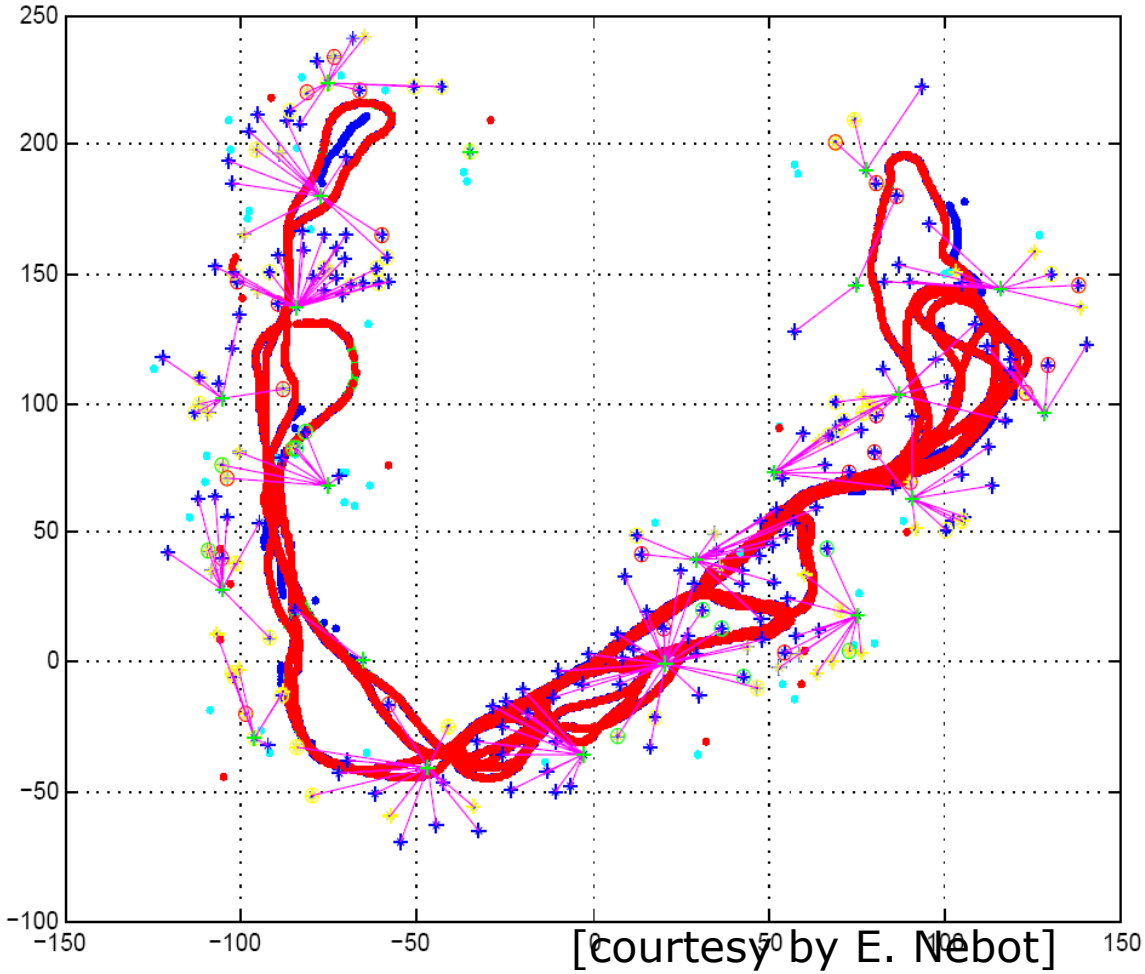
[Dissanayake et al., 2001]

EKF SLAM Example: Victoria Park Dataset

Victoria Park: Data Acquisition

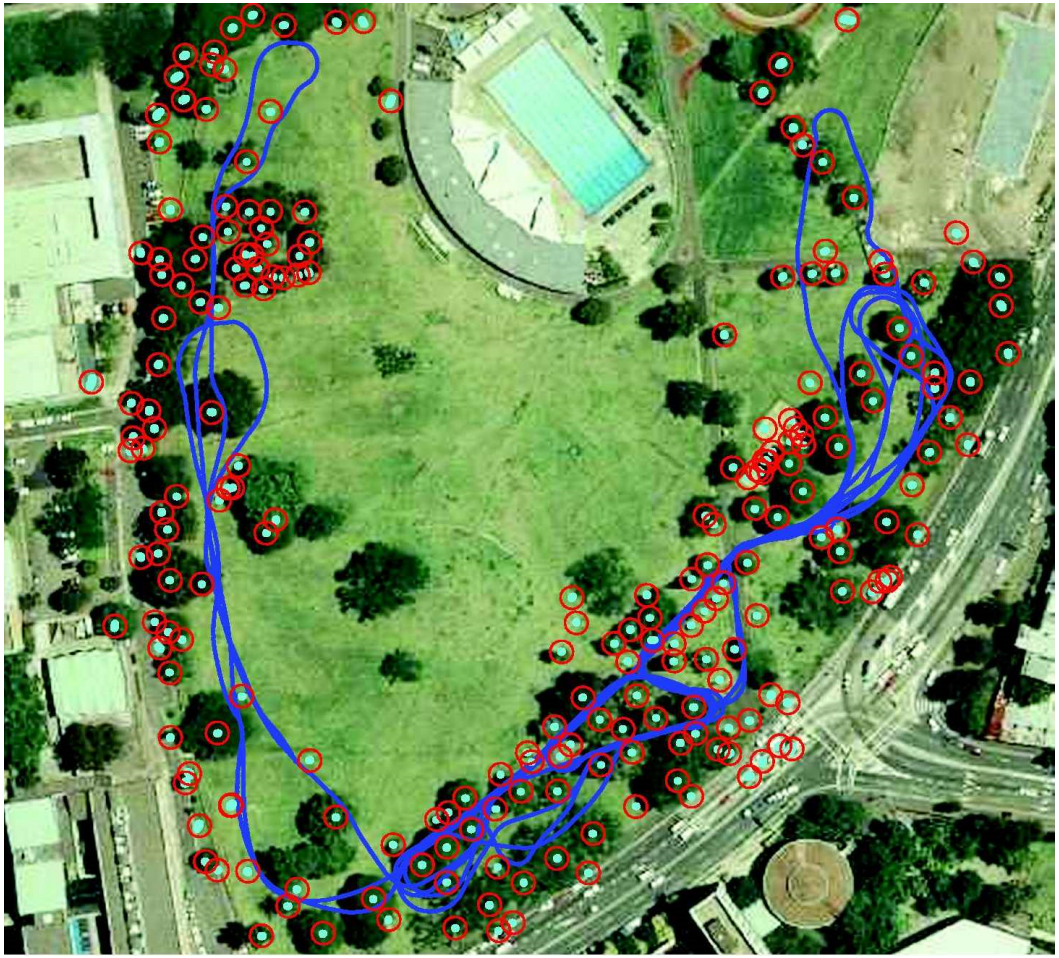
[courtesy by E. Nebot]

Victoria Park: Estimated Trajectory



[courtesy by E. Nebot]

Victoria Park: Landmarks

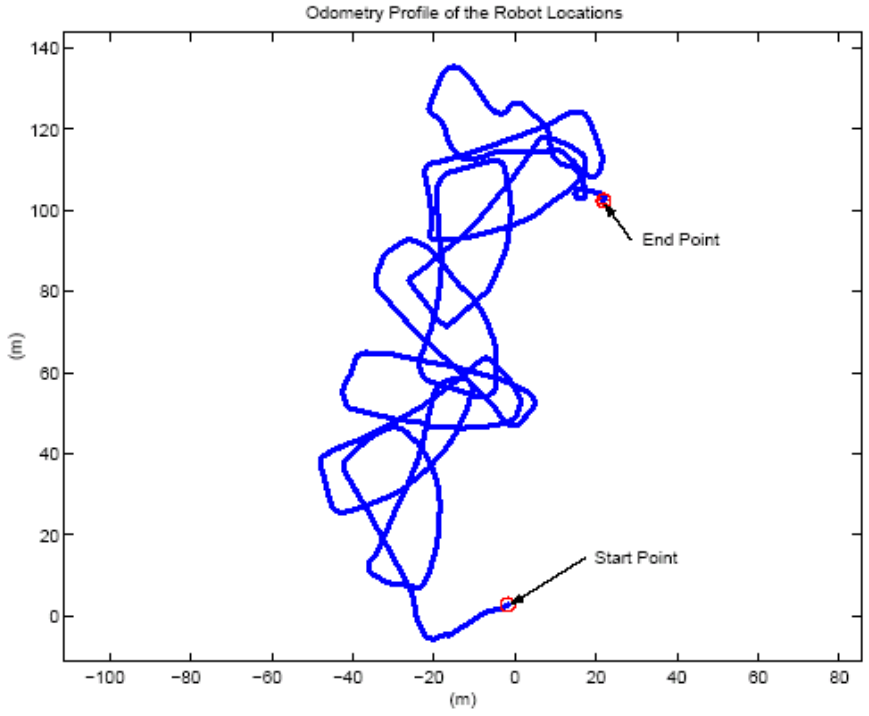


[courtesy by E. Nebot]

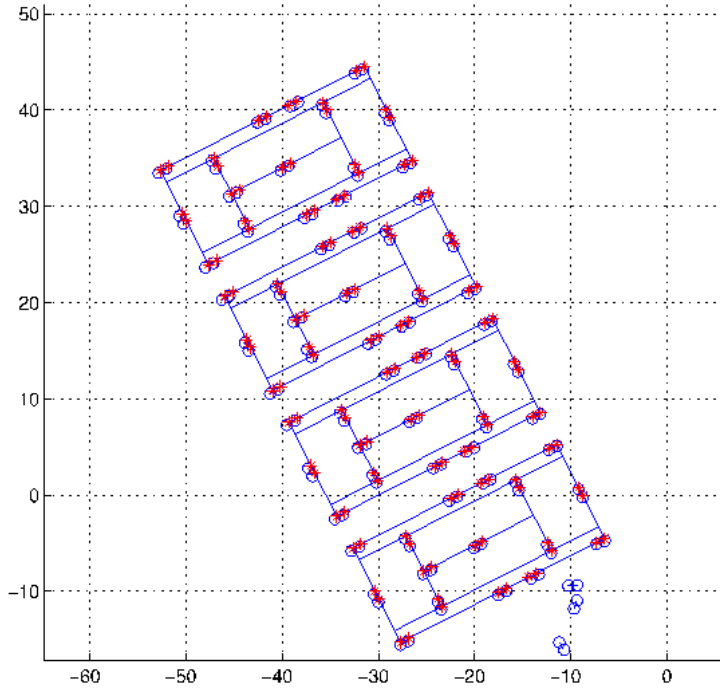
EKF SLAM Example: Tennis Court

[courtesy by J. Leonard]

EKF SLAM Example: Tennis Court



odometry

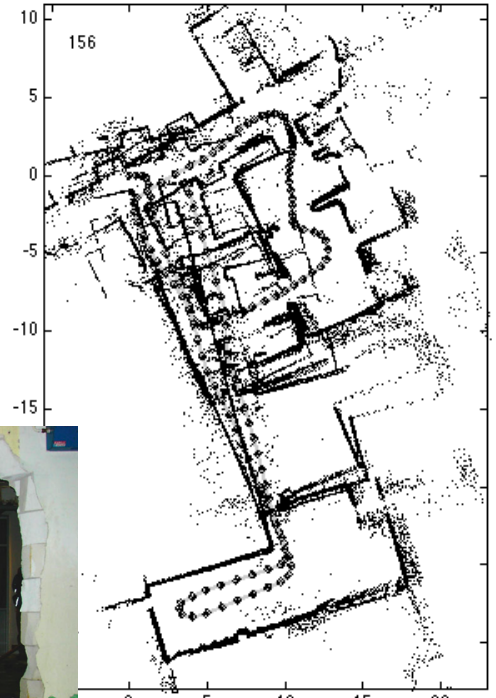


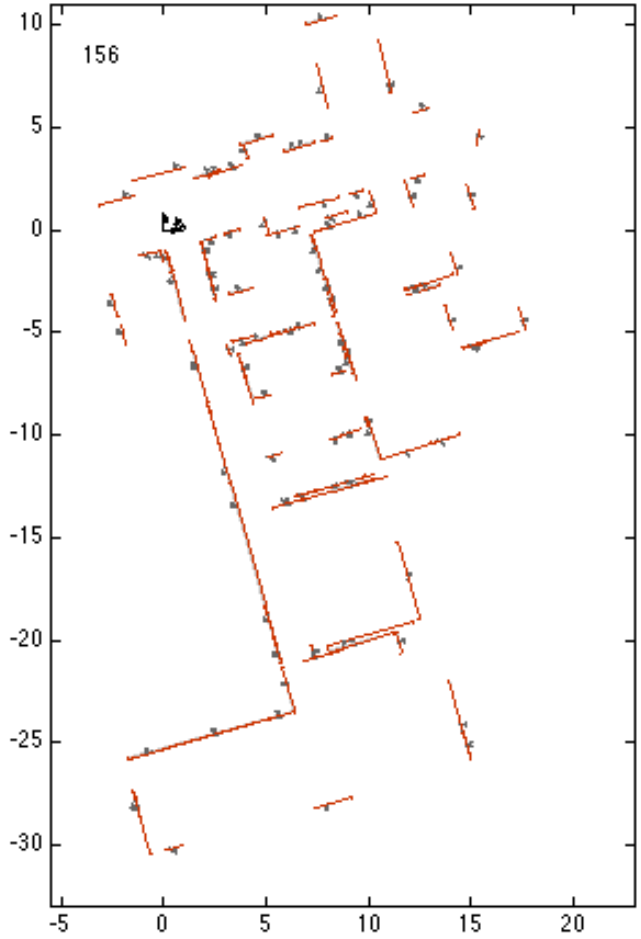
estimated trajectory

[courtesy by John Leonard]

EKF SLAM Example: Line Features

- KTH Bakery Data Set





[Wulf et al., ICRA 04]

EKF-SLAM: Complexity

- Cost per step: quadratic in n, the number of landmarks: $O(n^2)$
- Total cost to build a map with n landmarks: $O(n^3)$
- Memory consumption: $O(n^2)$
- Problem: becomes computationally intractable for large maps!
- There exists variants to circumvent these problems

SLAM Techniques

- EKF SLAM
- FastSLAM
- Graph-based SLAM
- Topological SLAM
  (mainly place recognition)
- Scan Matching / Visual Odometry
  (only locally consistent maps)
- Approximations for SLAM: Local submaps, Sparse extended information filters, Sparse links, Thin junction tree filters, etc.
- …

EKF-SLAM: Summary

- The first SLAM solution
- Convergence proof for linear Gaussian case
- Can diverge if nonlinearities are large (and the reality is nonlinear...)
- Can deal only with a single mode
- Successful in medium-scale scenes
- Approximations exists to reduce the computational complexity

Bayesian Methods

What do we know at time t?

1. A prior distribution $p(x_0)$
2. A history of measurements, $\mathbf{y}_{0:t}$
3. A history of inputs, $\mathbf{u}_{0:t-1}$

What we don't know is the actual sequence of states that the robot experienced.

Bayesian Methods

What do we know at time t?

1. A prior distribution $p(x_0)$
2. A history of measurements, $\mathbf{y}_{0:t}$
3. A history of inputs, $\mathbf{u}_{0:t-1}$

What we don't know is the actual sequence of states that the robot experienced.

Instead, we consider the conditional probability distribution over trajectories given the measurements we obtained and the control actions we took:

$$p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}).$$

Bayesian Methods

What do we know at time t?

1. A prior distribution $p(x_0)$
2. A history of measurements, $\mathbf{y}_{0:t}$
3. A history of inputs, $\mathbf{u}_{0:t-1}$

What we don't know is the actual sequence of states that the robot experienced.

Instead, we consider the conditional probability distribution over trajectories given the measurements we obtained and the control actions we took:

$$p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}).$$

Recursive Bayesian Estimation

The distribution $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})$ can be complicated.

It may be computed recursively when the state x_t is a sufficient statistic at each t.

In this case, the conditional distribution $p(x_t|\mathbf{x}_{0:t-1}, \mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})$ is equivalent to $p(x_t|x_{t-1}, u_t)$.

This distribution captures the dynamics at time t, which are Markovian.

Recursive Bayesian Estimation

The distribution $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})$ can be complicated.

It may be computed recursively when the state x_t is a sufficient statistic at each t.

In this case, the conditional distribution $p(x_t|\mathbf{x}_{0:t-1}, \mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})$ is equivalent to $p(x_t|x_{t-1}, u_t)$.

This distribution captures the dynamics at time t, which are Markovian.

The conditional distribution $p(y_t|\mathbf{x}_{0:t-1}, \mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})$ is equivalent to $p(y_t|x_t)$.

This distribution describes the measurement model.

Recursive Bayesian Estimation

The distribution $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})$ can be complicated.

It may be computed recursively when the state x_t is a sufficient statistic at each t.

In this case, the conditional distribution $p(x_t|\mathbf{x}_{0:t-1}, \mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})$ is equivalent to $p(x_t|x_{t-1}, u_t)$.

This distribution captures the dynamics at time t, which are Markovian.

The conditional distribution $p(y_t|\mathbf{x}_{0:t-1}, \mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})$ is equivalent to $p(y_t|x_t)$.

This distribution describes the measurement model.

We often solve problems *as if* our system is Markovian, even though we don't really know if that's true

Bayes Filter

Our estimate of the state x at time t is captured by the distribution

$$\mathrm{bel}(x_t) = p(x_t|\mathbf{y}_{1:t}, \mathbf{u}_{1:t}),$$

where we assume this distribution at time t is computed after u_t is taken and y_t measured. This distribution is our *belief* about the state x_t.

Bayes' rule allows us to derive

$$\mathrm{bel}(x_t) = \eta p(y_t|x_t) \int p(x_t|x_{t-1}, u_t)\mathrm{bel}(x_{t-1})dx_{t-1}$$

Bayes Filter: Derivation

$$
\begin{aligned}
p(x_t|y_t, \mathbf{y}_{1:t-1}, u_{1:t}) &= \frac{p(y_t|x_t, \mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})p(x_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})}{p(y_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})} \\
&= \frac{p(y_t|x_t)p(x_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})}{p(y_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})} \\
&= \eta p(y_t|x_t)p(x_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}) \\
&= \eta p(y_t|x_t)p(x_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}) \\
&= \eta p(y_t|x_t) \int p(x_t|u_t, x_{t-1})p(x_{t-1}|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t-1})dx \\
\mathrm{bel}(x_t) &= \eta p(y_t|x_t) \int p(x_t|u_t, x_{t-1})\mathrm{bel}(x_{t-1})dx
\end{aligned}
$$

Bayes Filter: Alternate Derivation

We have a model $p(x_t|x_{t-1}, u_t)$. If we know x_{t-1}, u_t, then we could set our motion-based update as $\overline{\text{bel}}(x_t) = p(x_t|x_{t-1}, u_t)$

When we have uncertainty in x_{t-1}, corresponding to $\text{bel}(x_{t-1})$, we instead calculate

$$\overline{\text{bel}}(x_t) = \int p(x_t|x_{t-1}, u_t)\text{bel}(x_{t-1})dx_{t-1}$$

Bayes Filter: Alternate Derivation

We have a model $p(x_t|x_{t-1}, u_t)$. If we know x_{t-1}, u_t, then we could set our motion-based update as $\overline{\text{bel}}(x_t) = p(x_t|x_{t-1}, u_t)$

When we have uncertainty in x_{t-1}, corresponding to $\text{bel}(x_{t-1})$, we instead calculate

$$\overline{\text{bel}}(x_t) = \int p(x_t|x_{t-1}, u_t)\text{bel}(x_{t-1})dx_{t-1}$$

$\overline{\text{bel}}(x_t)$ is our new belief distribution for x_t We can then apply Bayes' rule to the observation y_t:

$$p(x_t|y_t) = \frac{p(y_t|x_t)p(x_t)}{p(y_t)} = \eta p(y_t|x_t)\overline{\text{bel}}(x_t)$$

Kalman Filter

The Kalman Filter was developed for discrete-time linear time-invariant dynamical systems driven by noise:

$$x_{t+1} = Ax_t + v_t; \quad y_t = Cx_t + w_t,$$

where v_t and w_t represent zero-mean gaussian noise with covariances V and W respectively.

Kalman Filter

The Kalman Filter was developed for discrete-time linear time-invariant dynamical systems driven by noise:

$$x_{t+1} = Ax_t + v_t; \quad y_t = Cx_t + w_t,$$

where v_t and w_t represent zero-mean gaussian noise with covariances V and W respectively.

The first equation is our stochastic dynamics model; the second is our noisy sensor model

Kalman Filter

The Kalman Filter was developed for discrete-time linear time-invariant dynamical systems driven by noise:

$$x_{t+1} = Ax_t + v_t; \quad y_t = Cx_t + w_t,$$

where v_t and w_t represent zero-mean gaussian noise with covariances V and W respectively.

The first equation is our stochastic dynamics model; the second is our noisy sensor model

The Bayesian update

$$\mathrm{bel}(x_t) = \eta p(y_t|x_t) \int p(x_t|x_{t-1}, u_t)\mathrm{bel}(x_{t-1})dx_{t-1}$$

are easier to handle for this type of system, so that the Kalman Filter is often taught to control theorists without informing them about Bayes' existence.

Kalman Filter

We represent $\mathrm{bel}(x_t)$ using a mean $\mu_t = \mathbb{E}[x_t]$ and error covariance $\Sigma_t = \mathbb{E}\left[(x_{t+1} - \mu_t)(x_{t+1} - \mu_t)^T\right]$.

Kalman Filter

We represent $\mathrm{bel}(x_t)$ using a mean $\mu_t = \mathbb{E}[x_t]$ and error covariance $\Sigma_t = \mathbb{E}\left[(x_{t+1} - \mu_t)(x_{t+1} - \mu_t)^T\right]$.

If $x_{t+1}^{pred} = Ax_t + v_t$, then

$$\begin{aligned}\mu_{t+1}^{pred} &= \mathbb{E}[x_{t+1}] = \mathbb{E}[Ax_t + v_t] \\ &= A\mathbb{E}[x_t] + \mathbb{E}[v_t] \\ &= A\mu_t\end{aligned}$$

The predicted error covariance is then

$$\begin{aligned}\Sigma_{t+1}^{pred} &= \mathbb{E}\left[(x_{t+1} - \mu_{t+1}^{pred})(x_{t+1} - \mu_{t+1}^{pred})^T\right] \\ &= \mathbb{E}\left[(Ax_t + v_t - A\mu_t)(Ax_t + v_t - A\mu_t)^T\right] \\ &= A\Sigma_t A^T + V\end{aligned}$$

Kalman Filter

Once we get the measurement y_{t+1}, we then correct the predicted mean as

$$\mu_{t+1} = \mu_{t+1}^{pred} + L(y_{t+1} - C\mu_{t+1}^{pred}) = (I - LC)\mu_{t+1}^{pred} + Ly_{t+1} = M\mu_{t+1} + Ly_{t+1},$$

where $M = I - LC$.

Kalman Filter

Once we get the measurement y_{t+1}, we then correct the predicted mean as

$$\mu_{t+1} = \mu_{t+1}^{pred} + L(y_{t+1} - C\mu_{t+1}^{pred}) = (I - LC)\mu_{t+1}^{pred} + Ly_{t+1} = M\mu_{t+1} + Ly_{t+1},$$

where $M = I - LC$.

Note that $y_{t+1} = Cx_{t+1} + w_{t+1}$. The error covariance at $t+1$ is

$$\begin{aligned}
\Sigma_{t+1} &= \mathbb{E}\left[(x_{t+1} - \mu_{t+1})(x_{t+1} - \mu_{t+1})^T\right] \\
&= \mathbb{E}\left[\left(x_{t+1} - M\mu_{t+1}^{pred} - Ly_{t+1}\right)\left(x_{t+1} - M\mu_{t+1}^{pred} - Ly_{t+1}\right)^T\right] \\
&= \mathbb{E}\left[\left(M(x_{t+1} - \mu_{t+1}^{pred}) - Lw_{t+1}\right)\left(M(x_{t+1} - \mu_{t+1}^{pred})\right.\right. \\
&\qquad\qquad \left.\left. - Lw_{t+1}\right)^T\right]
\end{aligned}$$

Final Error Covariance

$$
\begin{aligned}
\Sigma_{t+1} &= \mathbb{E}\left[(x_{t+1} - \mu_{t+1})(x_{t+1} - \mu_{t+1})^T\right] \\
&= \mathbb{E}\left[\left(x_{t+1} - M\mu_{t+1}^{pred} - Ly_{t+1}\right)\left(x_{t+1} - M\mu_{t+1}^{pred} - Ly_{t+1}\right)^T\right] \\
&= \mathbb{E}\left[\left(M(x_{t+1} - \mu_{t+1}^{pred})\right)\left(M(x_{t+1} - \mu_{t+1}^{pred})\right)^T\right] \\
&\quad + \mathbb{E}\left[Lw_{t+1}(Lw_{t+1})^T\right] \\
&= \mathbb{E}\left[\left(M(x_{t+1} - \mu_{t+1}^{pred})\right)\left(M(x_{t+1} - \mu_{t+1}^{pred})\right)^T\right] \\
&\quad + \mathbb{E}\left[Lw_{t+1}(Lw_{t+1})^T\right] \\
&= M\mathbb{E}\left[\left((x_{t+1} - \mu_{t+1}^{pred})\right)\left((x_{t+1} - \mu_{t+1}^{pred})\right)^T\right]M^T \\
&\quad + L\mathbb{E}\left[w_{t+1}(w_{t+1})^T\right]L^T \\
&= (I - LC)\left(A\Sigma_t A^T + V\right)(I - LC)^T + LWL^T
\end{aligned}
$$

Kalman Filter Design

The uncertainty in the error depends on our choice of the Kalman Gain L.

The optimal choice of L minimizes the expected norm of the error, which is equivalent to minimizing the trace of Σ_{t+1}.

The gradient of the trace of Σ_{t+1} with respect to L becomes

$$-2(C(A\Sigma_t A + V))^T + 2L\left(C\left(A\Sigma_t A + V\right)C^T + W\right),$$

so that the optimal Kalman gain is

$$L = (A\Sigma_t A + V)C^T\left(C\left(A\Sigma_t A + V\right)C^T + W\right)^{-1}$$

At which point the covariance Σ_{t+1} becomes

$$\Sigma_{t+1} = (I - LC)(A\Sigma_t A + V)$$

Particle Filter

The Kalman filter is a Bayes filter that exploits the linear structure in the dynamics and measurement model.

It also relies on all distributions remaining Gaussian. [cf. Conjugate priors]

Particle Filter

The Kalman filter is a Bayes filter that exploits the linear structure in the dynamics and measurement model.

It also relies on all distributions remaining Gaussian. [cf. Conjugate priors]

For the case where the posterior distribution is difficult to describe using a parametric distribution, particle filters are a widely used alternative.

Particle Filter

The Kalman filter is a Bayes filter that exploits the linear structure in the dynamics and measurement model.

It also relies on all distributions remaining Gaussian. [cf. Conjugate priors]

For the case where the posterior distribution is difficult to describe using a parametric distribution, particle filters are a widely used alternative.

The distribution is represented by a set of samples independently drawn from the distribution.

Particle Filter: Algorithm

1. Move your samples according to the motion model.
2. Weight samples based on how likely they explain the measurement
3. Use the weights to resample from the existing set of points.
4. Return the resampled points as the new set of samples.

Luenberger Observer

The general idea is:

Luenberger Observer

The general idea is:

We're given models $\dot{x} = f(x, u)$, $y = h(x)$.

Luenberger Observer

The general idea is:

We're given models $\dot{x} = f(x, u)$, $y = h(x)$.

Simulate a copied version on your computer, using a parameter L:

$$\frac{d}{dt}\hat{x} = f(\hat{x}, u) + L(y - h(\hat{x}))$$

Luenberger Observer

The general idea is:

We're given models $\dot{x} = f(x, u)$, $y = h(x)$.

Simulate a copied version on your computer, using a parameter L:

$$\frac{d}{dt}\hat{x} = f(\hat{x}, u) + L(y - h(\hat{x}))$$

Our estimation involves 'correcting' the model-based dynamics.

The correction term is the error between reality and the prediction.

Luenberger Observer

The general idea is:

We're given models $\dot{x} = f(x, u)$, $y = h(x)$.

Simulate a copied version on your computer, using a parameter L:

$$\frac{d}{dt}\hat{x} = f(\hat{x}, u) + L(y - h(\hat{x}))$$

Our estimation involves 'correcting' the model-based dynamics.

The correction term is the error between reality and the prediction.

Obstacle

How do we choose L? How do we initialize our estimate?

Luenberger Observer: Analysis

$$\text{L. Obs, :} \qquad \frac{d}{dt}\hat{x} = f(\hat{x}, u) + L(y - h(\hat{x}))$$

Good estimation (according to control theorists) occurs when

$$\hat{x}(t) \to x(t) \text{ as } t \to \infty.$$

This property is the same as saying that the origin of system

$$\frac{d}{dt}e(t) = \frac{d}{dt}x(t) - \frac{d}{dt}\hat{h}(t)$$

be asymptotically stable, where $e(t) = x(t) - \hat{x}(t)$.

Luenberger Observer: Analysis

$$\begin{aligned}\frac{d}{dt}e(t) &= \frac{d}{dt}x(t) - \frac{d}{dt}\hat{h}(t) \\ &= f(x,u) - \hat{f}(x,u) - L(y - h(\hat{x})) \\ &= f(x,u) - \hat{f}(x,u) - L(h(x) - h(\hat{x}))\end{aligned}$$

It's hard to analyze the evolution of $e(t)$ for generic f and h.

Luenberger Observer: Analysis

$$\begin{aligned}\frac{d}{dt}e(t) &= \frac{d}{dt}x(t) - \frac{d}{dt}\hat{h}(t) \\ &= f(x,u) - \hat{f}(x,u) - L(y - h(\hat{x})) \\ &= f(x,u) - \hat{f}(x,u) - L(h(x) - h(\hat{x}))\end{aligned}$$

It's hard to analyze the evolution of $e(t)$ for generic f and h.

However, if our system is linear, we get

$$\begin{aligned}\frac{d}{dt}e(t) &= f(x,u) - \hat{f}(x,u) - L(h(x) - h(\hat{x})) \\ &= Ax + Bu - (A\hat{x} + Bu) + L(Cx - C\hat{x}) \\ &= Ax - A\hat{x} + LC(x - \hat{x}) \\ &= (A - LC)(x - \hat{x}) \\ &= (A - LC)e(t)\end{aligned}$$

Luenberger Observer: Analysis

$$\begin{aligned}\frac{d}{dt}e(t) &= \frac{d}{dt}x(t) - \frac{d}{dt}\hat{h}(t) \\ &= f(x,u) - \hat{f}(x,u) - L(y - h(\hat{x})) \\ &= f(x,u) - \hat{f}(x,u) - L(h(x) - h(\hat{x}))\end{aligned}$$

It's hard to analyze the evolution of $e(t)$ for generic f and h.

However, if our system is linear, we get

$$\begin{aligned}\frac{d}{dt}e(t) &= f(x,u) - \hat{f}(x,u) - L(h(x) - h(\hat{x})) \\ &= Ax + Bu - (A\hat{x} + Bu) + L(Cx - C\hat{x}) \\ &= Ax - A\hat{x} + LC(x - \hat{x}) \\ &= (A - LC)(x - \hat{x}) \\ &= (A - LC)e(t)\end{aligned}$$

Luenberger Observer: Design

$$\frac{d}{dt}e(t) = (A - LC)e(t)$$

We just need $(A - LC)$ to be Hurwitz.

Luenberger Observer: Design

$$\frac{d}{dt}e(t) = (A - LC)e(t)$$

We just need $(A - LC)$ to be Hurwitz.

We then implement the observer

$$\frac{d}{dt}\hat{x} = A\hat{x} + Bu + L(y - C\hat{x}).$$

Luenberger Observer: Design

$$\frac{d}{dt}e(t) = (A - LC)e(t)$$

We just need $(A - LC)$ to be Hurwitz.

We then implement the observer

$$\frac{d}{dt}\hat{x} = A\hat{x} + Bu + L(y - C\hat{x}).$$

If our system is nonlinear, we may be able to linearize to design L and use in an observer, provided $e(t)$ is small enough:

$$\frac{d}{dt}\hat{x} = f(\hat{x}, u) + L(y - h(\hat{x})).$$

Luenberger Observer: Issues

1. Q: What control u do we use?
   - For asymptotic stability, $u = -K\hat{x}$ works when $(A - BK)$ is Hurwitz
   - If we are optimizing a quadratic objective, we get the Linear Quadratic Gaussian problem, with solution $u = -K(t)\hat{x}(t)$.
   - Both cases have a separation principle: can design K and L separately.
2. For nonlinear systems handled by linearization, we need $e(t)$ to be small enough for the L to work. Determining how small is small enough is a separate issue.