Localization: Part 7
The Extended Kalman Filter

Computer Science 6912

Department of Computer Science
Memorial University of Newfoundland

July 4, 2018

The Extended Kalman Filter

- The assumption that the next state is a linear function of the previous state is often invalid (observation function also typically non-linear)

The Extended Kalman Filter

- The assumption that the next state is a linear function of the previous state is often invalid (observation function also typically non-linear)
- The Extended Kalman Filter (EKF) finds linear approximations to these non-linear functions

The Extended Kalman Filter

- The assumption that the next state is a linear function of the previous state is often invalid (observation function also typically non-linear)
- The Extended Kalman Filter (EKF) finds linear approximations to these non-linear functions
- First consider the impact on a Gaussian distributed random variable X passing through a function:

The Extended Kalman Filter

- The assumption that the next state is a linear function of the previous state is often invalid (observation function also typically non-linear)
- The Extended Kalman Filter (EKF) finds linear approximations to these non-linear functions
- First consider the impact on a Gaussian distributed random variable X passing through a function:
  - Linear function: $y = ax + b$

The Extended Kalman Filter

- The assumption that the next state is a linear function of the previous state is often invalid (observation function also typically non-linear)
- The Extended Kalman Filter (EKF) finds linear approximations to these non-linear functions
- First consider the impact on a Gaussian distributed random variable X passing through a function:
  - Linear function: $y = ax + b$
    - The resulting r.v. Y has mean $a\mu + b$ and variance $a^2\sigma^2$

The Extended Kalman Filter

- The assumption that the next state is a linear function of the previous state is often invalid (observation function also typically non-linear)
- The Extended Kalman Filter (EKF) finds linear approximations to these non-linear functions
- First consider the impact on a Gaussian distributed random variable X passing through a function:
  - Linear function: $y = ax + b$
    - The resulting r.v. Y has mean $a\mu + b$ and variance $a^2\sigma^2$
    - Part (a) of the figure on the next page

The Extended Kalman Filter

- The assumption that the next state is a linear function of the previous state is often invalid (observation function also typically non-linear)
- The Extended Kalman Filter (EKF) finds linear approximations to these non-linear functions
- First consider the impact on a Gaussian distributed random variable X passing through a function:
  - Linear function: $y = ax + b$
    - The resulting r.v. Y has mean $a\mu + b$ and variance $a^2\sigma^2$
    - Part (a) of the figure on the next page
  - Non-linear function: $y = g(x)$

The Extended Kalman Filter

- The assumption that the next state is a linear function of the previous state is often invalid (observation function also typically non-linear)
- The Extended Kalman Filter (EKF) finds linear approximations to these non-linear functions
- First consider the impact on a Gaussian distributed random variable X passing through a function:
  - Linear function: $y = ax + b$
    - The resulting r.v. Y has mean $a\mu + b$ and variance $a^2\sigma^2$
    - Part (a) of the figure on the next page
  - Non-linear function: $y = g(x)$
    - Distribution no longer Gaussian!

The Extended Kalman Filter

- The assumption that the next state is a linear function of the previous state is often invalid (observation function also typically non-linear)
- The Extended Kalman Filter (EKF) finds linear approximations to these non-linear functions
- First consider the impact on a Gaussian distributed random variable X passing through a function:
  - Linear function: $y = ax + b$
    - The resulting r.v. Y has mean $a\mu + b$ and variance $a^2\sigma^2$
    - Part (a) of the figure on the next page
  - Non-linear function: $y = g(x)$
    - Distribution no longer Gaussian!
    - Part (b) of the figure on the next page

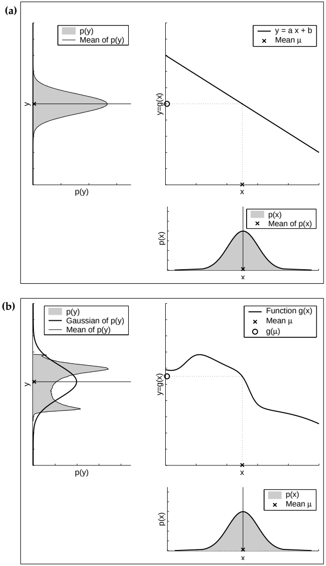


Figure 3.3 (a) Linear and (b) nonlinear transformation of a Gaussian random variable. The lower right plots show the density of the original random variable, X. This random variable is passed through the function displayed in the upper right graphs (the transformation of the mean is indicated by the dotted line). The density of the resulting random variable Y is plotted in the upper left graphs.

- We assume that the system state update and measurement equations are non-linear,

- We assume that the system state update and measurement equations are non-linear,

- We assume that the system state update and measurement equations are non-linear,

$$x_t = g(u_t, x_{t-1}) + \epsilon_t$$

- We assume that the system state update and measurement equations are non-linear,

$$
\begin{aligned}
x_t &= g(u_t, x_{t-1}) + \epsilon_t \\
z_t &= h(x_t) + \delta_t
\end{aligned}
$$

- We assume that the system state update and measurement equations are non-linear,

$$\begin{aligned} x_t &= g(u_t, x_{t-1}) + \epsilon_t \\ z_t &= h(x_t) + \delta_t \end{aligned}$$

- We will apply first-order Taylor series approximations to the functions g and h

- We assume that the system state update and measurement equations are non-linear,

$$
\begin{aligned}
x_t &= g(u_t, x_{t-1}) + \epsilon_t \\
z_t &= h(x_t) + \delta_t
\end{aligned}
$$

- We will apply first-order Taylor series approximations to the functions g and h

- We assume that the system state update and measurement equations are non-linear,

$$\begin{aligned} x_t &= g(u_t, x_{t-1}) + \epsilon_t \\ z_t &= h(x_t) + \delta_t \end{aligned}$$

- We will apply first-order Taylor series approximations to the functions g and h; These will be linear equations

- We assume that the system state update and measurement equations are non-linear,

$$
\begin{aligned}
x_t &= g(u_t, x_{t-1}) + \epsilon_t \\
z_t &= h(x_t) + \delta_t
\end{aligned}
$$

- We will apply first-order Taylor series approximations to the functions g and h; These will be linear equations
- First, recall the Taylor series

- We assume that the system state update and measurement equations are non-linear,

$$
\begin{aligned}
x_t &= g(u_t, x_{t-1}) + \epsilon_t \\
z_t &= h(x_t) + \delta_t
\end{aligned}
$$

- We will apply first-order Taylor series approximations to the functions g and h; These will be linear equations
- First, recall the Taylor series

- We assume that the system state update and measurement equations are non-linear,

$$\begin{aligned} x_t &= g(u_t, x_{t-1}) + \epsilon_t \\ z_t &= h(x_t) + \delta_t \end{aligned}$$

- We will apply first-order Taylor series approximations to the functions g and h; These will be linear equations
- First, recall the Taylor series

$$f(x) = f(c) + f'(c)(x - c) + \frac{1}{2!} f''(c)(x - c)^2 + \cdots$$

- We assume that the system state update and measurement equations are non-linear,

$$\begin{aligned} x_t &= g(u_t, x_{t-1}) + \epsilon_t \\ z_t &= h(x_t) + \delta_t \end{aligned}$$

- We will apply first-order Taylor series approximations to the functions g and h; These will be linear equations
- First, recall the Taylor series

$$f(x) = f(c) + f'(c)(x - c) + \frac{1}{2!} f''(c)(x - c)^2 + \cdots$$

The Taylor series gives you $f(x)$ when you only know $f(c)$, derivatives of f at c, and $(x - c)$

- We assume that the system state update and measurement equations are non-linear,

$$\begin{aligned} x_t &= g(u_t, x_{t-1}) + \epsilon_t \\ z_t &= h(x_t) + \delta_t \end{aligned}$$

- We will apply first-order Taylor series approximations to the functions g and h; These will be linear equations
- First, recall the Taylor series

$$f(x) = f(c) + f'(c)(x-c) + \frac{1}{2!}f''(c)(x-c)^2 + \cdots$$

The Taylor series gives you $f(x)$ when you only know $f(c)$, derivatives of f at c, and $(x-c)$
- A first-order Taylor series approximation of $g(u_t, x_{t-1})$,

- We assume that the system state update and measurement equations are non-linear,

$$
\begin{aligned}
x_t &= g(u_t, x_{t-1}) + \epsilon_t \\
z_t &= h(x_t) + \delta_t
\end{aligned}
$$

- We will apply first-order Taylor series approximations to the functions g and h; These will be linear equations
- First, recall the Taylor series

$$
f(x) = f(c) + f'(c)(x-c) + \frac{1}{2!}f''(c)(x-c)^2 + \cdots
$$

  The Taylor series gives you $f(x)$ when you only know $f(c)$, derivatives of f at c, and $(x-c)$
- A first-order Taylor series approximation of $g(u_t, x_{t-1})$,

- We assume that the system state update and measurement equations are non-linear,

$$\begin{aligned} x_t &= g(u_t, x_{t-1}) + \epsilon_t \\ z_t &= h(x_t) + \delta_t \end{aligned}$$

- We will apply first-order Taylor series approximations to the functions g and h; These will be linear equations
- First, recall the Taylor series

$$f(x) = f(c) + f'(c)(x-c) + \frac{1}{2!} f''(c)(x-c)^2 + \cdots$$

The Taylor series gives you $f(x)$ when you only know $f(c)$, derivatives of f at c, and $(x - c)$

- A first-order Taylor series approximation of $g(u_t, x_{t-1})$,

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})\ (x_{t-1} - \mu_{t-1})$$

- We assume that the system state update and measurement equations are non-linear,

$$\begin{aligned} x_t &= g(u_t, x_{t-1}) + \epsilon_t \\ z_t &= h(x_t) + \delta_t \end{aligned}$$

- We will apply first-order Taylor series approximations to the functions g and h; These will be linear equations
- First, recall the Taylor series

$$f(x) = f(c) + f'(c)(x-c) + \frac{1}{2!} f''(c)(x-c)^2 + \cdots$$

  The Taylor series gives you $f(x)$ when you only know $f(c)$, derivatives of f at c, and $(x-c)$
- A first-order Taylor series approximation of $g(u_t, x_{t-1})$,

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})\ (x_{t-1} - \mu_{t-1})$$

- Approximation of g at x_{t-1} made at most likely value of x_{t-1}: μ_{t-1}

However, g is typically a vector function, hence we need the vector form of the Taylor series approximation,

However, g is typically a vector function, hence we need the vector form of the Taylor series approximation,

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t \ (x_{t-1} - \mu_{t-1})$$

However, g is typically a vector function, hence we need the vector form of the Taylor series approximation,

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t \; (x_{t-1} - \mu_{t-1})$$

G_t is the $n \times n$ **Jacobian** matrix of first derivatives of g

However, g is typically a vector function, hence we need the vector form of the Taylor series approximation,

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t \; (x_{t-1} - \mu_{t-1})$$

G_t is the $n \times n$ **Jacobian** matrix of first derivatives of g. What is this Jacobian matrix?

However, g is typically a vector function, hence we need the vector form of the Taylor series approximation,

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t \ (x_{t-1} - \mu_{t-1})$$

G_t is the $n \times n$ **Jacobian** matrix of first derivatives of g. What is this Jacobian matrix?

In general, if we have a function where the input is a vector, $\boldsymbol{x}$, of length n, and the output is a vector, $\boldsymbol{y}$, of length m, the Jacobian is,

However, g is typically a vector function, hence we need the vector form of the Taylor series approximation,

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t \ (x_{t-1} - \mu_{t-1})$$

G_t is the $n \times n$ **Jacobian** matrix of first derivatives of g. What is this Jacobian matrix?

In general, if we have a function where the input is a vector, $\boldsymbol{x}$, of length n, and the output is a vector, $\boldsymbol{y}$, of length m, the Jacobian is,

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

However, g is typically a vector function, hence we need the vector form of the Taylor series approximation,

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t \, (x_{t-1} - \mu_{t-1})$$

G_t is the $n \times n$ **Jacobian** matrix of first derivatives of g. What is this Jacobian matrix?

In general, if we have a function where the input is a vector, $\boldsymbol{x}$, of length n, and the output is a vector, $\boldsymbol{y}$, of length m, the Jacobian is,

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

You can think of the Jacobian as the derivative of a matrix.

- A linear approximation is also made to $h(x_t)$,

- A linear approximation is also made to $h(x_t)$,

- A linear approximation is also made to $h(x_t)$,

$$h(x_t) \approx h(\bar{\mu}_t) + h'(\bar{\mu}_t)\;(x_t - \bar{\mu}_t)$$

- A linear approximation is also made to $h(x_t)$,

$$h(x_t) \approx h(\bar{\mu}_t) + h'(\bar{\mu}_t)\ (x_t - \bar{\mu}_t)$$

- Approximation of h at x_t made at most likely value of x_t after the prediction step: $\bar{\mu}_t$

- A linear approximation is also made to $h(x_t)$,

$$h(x_t) \approx h(\bar{\mu}_t) + h'(\bar{\mu}_t)\ (x_t - \bar{\mu}_t)$$

- Approximation of h at x_t made at most likely value of x_t after the prediction step: $\bar{\mu}_t$
- Let $H_t = h'(\bar{\mu}_t)$

- A linear approximation is also made to $h(x_t)$,

$$h(x_t) \approx h(\bar{\mu}_t) + h'(\bar{\mu}_t)\ (x_t - \bar{\mu}_t)$$

- Approximation of h at x_t made at most likely value of x_t after the prediction step: $\bar{\mu}_t$
- Let $H_t = h'(\bar{\mu}_t)$
  - H_t is the $k \times n$ **Jacobian** matrix of first derivatives of h

The EKF Algorithm

1. EKF$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

The EKF Algorithm

1. $\text{EKF}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2. $\quad \bar{\mu}_t = g(u_t, \mu_{t-1})$

The EKF Algorithm

1. $\text{EKF}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2. $\quad \bar{\mu}_t = g(u_t, \mu_{t-1})$
3. $\quad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$

The EKF Algorithm

1. EKF($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2. $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3. $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$

The EKF Algorithm

1. EKF($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2. $\quad \bar{\mu}_t = g(u_t, \mu_{t-1})$
3. $\quad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $\quad K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5. $\quad \mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

The EKF Algorithm

1. $\text{EKF}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2. $\quad \bar{\mu}_t = g(u_t, \mu_{t-1})$
3. $\quad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $\quad K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5. $\quad \mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6. $\quad \Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

The EKF Algorithm

1. EKF($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2. $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3. $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6. $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
7. return μ_t, Σ_t

The EKF Algorithm

1. $\text{EKF}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2. $\quad \bar{\mu}_t = g(u_t, \mu_{t-1})$
3. $\quad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $\quad K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5. $\quad \mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6. $\quad \Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
7. $\quad$ return μ_t, Σ_t

Differences from Kalman filter algorithm:

The EKF Algorithm

1. EKF$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2. $\quad \bar{\mu}_t = g(u_t, \mu_{t-1})$
3. $\quad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $\quad K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5. $\quad \mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6. $\quad \Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
7. $\quad$ return μ_t, Σ_t

Differences from Kalman filter algorithm:

The Jacobian G_t replaces the matrices A_t and B_t

The EKF Algorithm

1. $\text{EKF}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2. $\quad \bar{\mu}_t = g(u_t, \mu_{t-1})$
3. $\quad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $\quad K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5. $\quad \mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6. $\quad \Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
7. $\quad$ return μ_t, Σ_t

Differences from Kalman filter algorithm:

The Jacobian G_t replaces the matrices A_t and B_t. The Jacobian H_t replaces C_t

The EKF Algorithm

1. $\text{EKF}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2. $\quad \bar{\mu}_t = g(u_t, \mu_{t-1})$
3. $\quad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $\quad K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5. $\quad \mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6. $\quad \Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
7. $\quad$ return μ_t, Σ_t

Differences from Kalman filter algorithm:

The Jacobian G_t replaces the matrices A_t and B_t. The Jacobian H_t replaces C_t. On line 2 the prediction is made directly through g

The EKF Algorithm

1. $\text{EKF}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2. $\quad \bar{\mu}_t = g(u_t, \mu_{t-1})$
3. $\quad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $\quad K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5. $\quad \mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6. $\quad \Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$
7. $\quad$ return μ_t, Σ_t

Differences from Kalman filter algorithm:

The Jacobian G_t replaces the matrices A_t and B_t. The Jacobian H_t replaces C_t. On line 2 the prediction is made directly through g. Similarly on line 5 the measurement prediction $C_t \bar{\mu}_t$ is replaced by $h(\bar{\mu}_t)$.

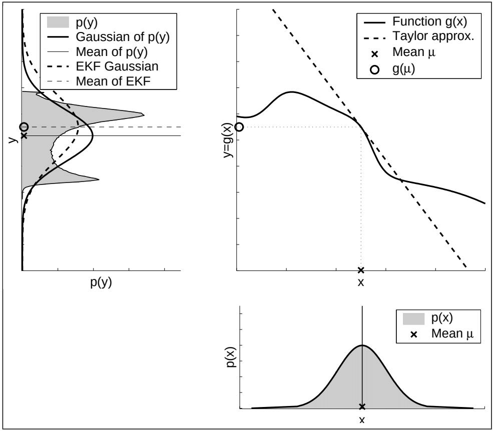


Figure 3.4 Illustration of linearization applied by the EKF. Instead of passing the Gaussian through the nonlinear function g, it is passed through a linear approximation of g. The linear function is tangent to g at the mean of the original Gaussian. The resulting Gaussian is shown as the dashed line in the upper left graph. The linearization incurs an approximation error, as indicated by the mismatch between the linearized Gaussian (dashed) and the Gaussian computed from the highly accurate Monte-Carlo estimate (solid).

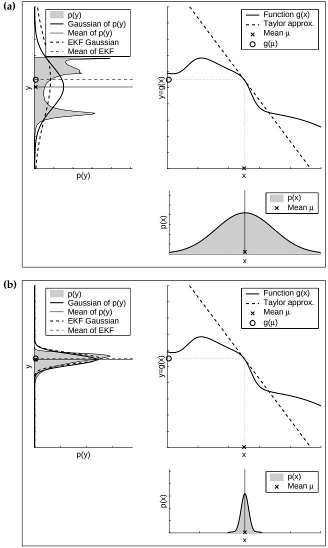


Figure 3.5 Dependency of approximation quality on uncertainty. Both Gaussians (lower right) have the same mean and are passed through the same nonlinear function (upper right). The higher uncertainty of the left Gaussian produces a more distorted density of the resulting random variable (gray area in upper left graph). The solid lines in the upper left graphs show the Gaussians extracted from these densities. The dashed lines represent the Gaussians generated by the linearization applied by the EKF.

Example: Differential Drive With No Sensors

In this example, we will consider only the prediction step of the EKF

Example: Differential Drive With No Sensors

In this example, we will consider only the prediction step of the EKF,

1. EKF_Prediction($\mu_{t-1}, \Sigma_{t-1}, u_t$)

Example: Differential Drive With No Sensors

In this example, we will consider only the prediction step of the EKF,

1. EKF_Prediction($\mu_{t-1}, \Sigma_{t-1}, u_t$)
2. $\quad \mu_t = g(u_t, \mu_{t-1})$

Example: Differential Drive With No Sensors

In this example, we will consider only the prediction step of the EKF,

1. EKF_Prediction($\mu_{t-1}, \Sigma_{t-1}, u_t$)
2. $\mu_t = g(u_t, \mu_{t-1})$
3. $\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$

Example: Differential Drive With No Sensors

In this example, we will consider only the prediction step of the EKF,

1. EKF_Prediction($\mu_{t-1}, \Sigma_{t-1}, u_t$)
2. $\quad \mu_t = g(u_t, \mu_{t-1})$
3. $\quad \Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $\quad$ return μ_t, Σ_t

Example: Differential Drive With No Sensors

In this example, we will consider only the prediction step of the EKF,

1. EKF_Prediction($\mu_{t-1}, \Sigma_{t-1}, u_t$)
2. $\mu_t = g(u_t, \mu_{t-1})$
3. $\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. return μ_t, Σ_t

The function $g(u_t, \mu_{t-1})$ comes from the forward kinematic equation for a D-D robot:

Example: Differential Drive With No Sensors

In this example, we will consider only the prediction step of the EKF,

1. EKF_Prediction($\mu_{t-1}, \Sigma_{t-1}, u_t$)
2. $\quad \mu_t = g(u_t, \mu_{t-1})$
3. $\quad \Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $\quad$ return μ_t, Σ_t

The function $g(u_t, \mu_{t-1})$ comes from the forward kinematic equation for a D-D robot:

$$\dot{x}_t = \dot{\xi}_I = \frac{r}{2} \begin{bmatrix} \cos(\theta)(\dot{\phi}_r + \dot{\phi}_l) \\ \sin(\theta)(\dot{\phi}_r + \dot{\phi}_l) \\ (\dot{\phi}_r - \dot{\phi}_l)/l \end{bmatrix}$$

Example: Differential Drive With No Sensors

In this example, we will consider only the prediction step of the EKF,

1. EKF_Prediction($\mu_{t-1}, \Sigma_{t-1}, u_t$)
2. $\quad \mu_t = g(u_t, \mu_{t-1})$
3. $\quad \Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. $\quad$ return μ_t, Σ_t

The function $g(u_t, \mu_{t-1})$ comes from the forward kinematic equation for a D-D robot:

$$\dot{x}_t = \dot{\xi}_I = \frac{r}{2} \begin{bmatrix} \cos(\theta)(\dot{\phi}_r + \dot{\phi}_l) \\ \sin(\theta)(\dot{\phi}_r + \dot{\phi}_l) \\ (\dot{\phi}_r - \dot{\phi}_l)/l \end{bmatrix}$$

The robot has some discrete update rate Δt

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$. We define,

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$. We define,

$$\Delta\theta = \frac{r(\Delta\phi_r - \Delta\phi_l)}{2l}$$

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$. We define,

$$\Delta\theta = \frac{r(\Delta\phi_r - \Delta\phi_l)}{2l} \qquad \Delta s = \frac{r(\Delta\phi_r + \Delta\phi_l)}{2}$$

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$. We define,

$$\Delta\theta = \frac{r(\Delta\phi_r - \Delta\phi_l)}{2l} \qquad \Delta s = \frac{r(\Delta\phi_r + \Delta\phi_l)}{2}$$

The forward kinematic equation for discrete updates is,

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$. We define,

$$\Delta\theta = \frac{r(\Delta\phi_r - \Delta\phi_l)}{2l} \qquad \Delta s = \frac{r(\Delta\phi_r + \Delta\phi_l)}{2}$$

The forward kinematic equation for discrete updates is,

$$x_t = x_{t-1} + \begin{bmatrix} \cos(\theta + \Delta\theta)\Delta s \\ \sin(\theta + \Delta\theta)\Delta s \\ \Delta\theta \end{bmatrix}$$

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$. We define,

$$\Delta\theta = \frac{r(\Delta\phi_r - \Delta\phi_l)}{2l} \qquad \Delta s = \frac{r(\Delta\phi_r + \Delta\phi_l)}{2}$$

The forward kinematic equation for discrete updates is,

$$x_t = x_{t-1} + \begin{bmatrix} \cos(\theta + \Delta\theta)\Delta s \\ \sin(\theta + \Delta\theta)\Delta s \\ \Delta\theta \end{bmatrix} = g(u_t, x_{t-1})$$

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$. We define,

$$\Delta\theta = \frac{r(\Delta\phi_r - \Delta\phi_l)}{2l} \qquad \Delta s = \frac{r(\Delta\phi_r + \Delta\phi_l)}{2}$$

The forward kinematic equation for discrete updates is,

$$x_t = x_{t-1} + \begin{bmatrix} \cos(\theta + \Delta\theta)\Delta s \\ \sin(\theta + \Delta\theta)\Delta s \\ \Delta\theta \end{bmatrix} = g(u_t, x_{t-1})$$

This equation is non-linear and does not fit the form required by the Kalman filter

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$. We define,

$$\Delta\theta = \frac{r(\Delta\phi_r - \Delta\phi_l)}{2l} \qquad \Delta s = \frac{r(\Delta\phi_r + \Delta\phi_l)}{2}$$

The forward kinematic equation for discrete updates is,

$$x_t = x_{t-1} + \begin{bmatrix} \cos(\theta + \Delta\theta)\Delta s \\ \sin(\theta + \Delta\theta)\Delta s \\ \Delta\theta \end{bmatrix} = g(u_t, x_{t-1})$$

This equation is non-linear and does not fit the form required by the Kalman filter. Step 2 of the EKF algorithm does not require any linearization

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$. We define,

$$\Delta\theta = \frac{r(\Delta\phi_r - \Delta\phi_l)}{2l} \qquad \Delta s = \frac{r(\Delta\phi_r + \Delta\phi_l)}{2}$$

The forward kinematic equation for discrete updates is,

$$x_t = x_{t-1} + \begin{bmatrix} \cos(\theta + \Delta\theta)\Delta s \\ \sin(\theta + \Delta\theta)\Delta s \\ \Delta\theta \end{bmatrix} = g(u_t, x_{t-1})$$

This equation is non-linear and does not fit the form required by the Kalman filter. Step 2 of the EKF algorithm does not require any linearization. Indeed, the equation above implements step 2 directly.

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta \phi_r$ and $\Delta \phi_l$. We define,

$$\Delta\theta = \frac{r(\Delta\phi_r - \Delta\phi_l)}{2l} \qquad \Delta s = \frac{r(\Delta\phi_r + \Delta\phi_l)}{2}$$

The forward kinematic equation for discrete updates is,

$$x_t = x_{t-1} + \begin{bmatrix} \cos(\theta + \Delta\theta)\Delta s \\ \sin(\theta + \Delta\theta)\Delta s \\ \Delta\theta \end{bmatrix} = g(u_t, x_{t-1})$$

This equation is non-linear and does not fit the form required by the Kalman filter. Step 2 of the EKF algorithm does not require any linearization. Indeed, the equation above implements step 2 directly.

Step 3 involves finding the Jacobian G_t and the covariance matrix R_t

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$. We define,

$$\Delta\theta = \frac{r(\Delta\phi_r - \Delta\phi_l)}{2l} \qquad \Delta s = \frac{r(\Delta\phi_r + \Delta\phi_l)}{2}$$

The forward kinematic equation for discrete updates is,

$$x_t = x_{t-1} + \begin{bmatrix} \cos(\theta + \Delta\theta)\Delta s \\ \sin(\theta + \Delta\theta)\Delta s \\ \Delta\theta \end{bmatrix} = g(u_t, x_{t-1})$$

This equation is non-linear and does not fit the form required by the Kalman filter. Step 2 of the EKF algorithm does not require any linearization. Indeed, the equation above implements step 2 directly.

Step 3 involves finding the Jacobian G_t and the covariance matrix R_t

3. $\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$

The robot has some discrete update rate Δt. In between updates, the two wheels will roll by $\Delta\phi_r$ and $\Delta\phi_l$. We define,

$$\Delta\theta = \frac{r(\Delta\phi_r - \Delta\phi_l)}{2l} \qquad \Delta s = \frac{r(\Delta\phi_r + \Delta\phi_l)}{2}$$

The forward kinematic equation for discrete updates is,

$$x_t = x_{t-1} + \begin{bmatrix} \cos(\theta + \Delta\theta)\Delta s \\ \sin(\theta + \Delta\theta)\Delta s \\ \Delta\theta \end{bmatrix} = g(u_t, x_{t-1})$$

This equation is non-linear and does not fit the form required by the Kalman filter. Step 2 of the EKF algorithm does not require any linearization. Indeed, the equation above implements step 2 directly.

Step 3 involves finding the Jacobian G_t and the covariance matrix R_t

3. $\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$

(step 3 again)

(step 3 again)

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$$

(step 3 again)

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$$

G_t is the Jacobian of g with respect to the input vector $\begin{bmatrix} x & y & \theta \end{bmatrix}$

(step 3 again)

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$$

G_t is the Jacobian of g with respect to the input vector $\begin{bmatrix} x & y & \theta \end{bmatrix}$. It is evaluated at μ_{t-1}, which is our current best guess for x_{t-1}

(step 3 again)

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$$

G_t is the Jacobian of g with respect to the input vector $\begin{bmatrix} x & y & \theta \end{bmatrix}$. It is evaluated at μ_{t-1}, which is our current best guess for x_{t-1}. It will be a reasonable approximation as long as g is not too non-linear, and as long as μ_{t-1} is reasonably close to the true x_{t-1}

(step 3 again)

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$$

G_t is the Jacobian of g with respect to the input vector $\begin{bmatrix} x & y & \theta \end{bmatrix}$. It is evaluated at μ_{t-1}, which is our current best guess for x_{t-1}. It will be a reasonable approximation as long as g is not too non-linear, and as long as μ_{t-1} is reasonably close to the true x_{t-1}.

$G_t \Sigma_{t-1} G_t^T$ models the propagation of uncertainty from the last iteration (see slide 7 from the notes on covariance matrices)

(step 3 again)

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$$

G_t is the Jacobian of g with respect to the input vector $\begin{bmatrix} x & y & \theta \end{bmatrix}$. It is evaluated at μ_{t-1}, which is our current best guess for x_{t-1}. It will be a reasonable approximation as long as g is not too non-linear, and as long as μ_{t-1} is reasonably close to the true x_{t-1}.

$G_t \Sigma_{t-1} G_t^T$ models the propagation of uncertainty from the last iteration (see slide 7 from the notes on covariance matrices).

$$G_t = \begin{bmatrix} \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} & \frac{\partial g}{\partial \theta} \end{bmatrix}$$

(step 3 again)

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$$

G_t is the Jacobian of g with respect to the input vector $\begin{bmatrix} x & y & \theta \end{bmatrix}$. It is evaluated at μ_{t-1}, which is our current best guess for x_{t-1}. It will be a reasonable approximation as long as g is not too non-linear, and as long as μ_{t-1} is reasonably close to the true x_{t-1}.

$G_t \Sigma_{t-1} G_t^T$ models the propagation of uncertainty from the last iteration (see slide 7 from the notes on covariance matrices).

$$\begin{aligned} G_t &= \begin{bmatrix} \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} & \frac{\partial g}{\partial \theta} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & -\sin(\theta + \Delta\theta)\Delta s \\ 0 & 1 & \cos(\theta + \Delta\theta)\Delta s \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

(step 3 again)

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + R_t$$

G_t is the Jacobian of g with respect to the input vector $\begin{bmatrix} x & y & \theta \end{bmatrix}$. It is evaluated at μ_{t-1}, which is our current best guess for x_{t-1}. It will be a reasonable approximation as long as g is not too non-linear, and as long as μ_{t-1} is reasonably close to the true x_{t-1}.

$G_t \Sigma_{t-1} G_t^T$ models the propagation of uncertainty from the last iteration (see slide 7 from the notes on covariance matrices).

$$\begin{aligned} G_t &= \begin{bmatrix} \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} & \frac{\partial g}{\partial \theta} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & -\sin(\theta + \Delta\theta)\Delta s \\ 0 & 1 & \cos(\theta + \Delta\theta)\Delta s \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

To compute R_t we will assume that the errors in $\Delta\phi_r$ and $\Delta\phi_l$ are independent

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

To compute R_t we will assume that the errors in $\Delta\phi_r$ and $\Delta\phi_l$ are independent. We define the following covariance matrix,

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

To compute R_t we will assume that the errors in $\Delta\phi_r$ and $\Delta\phi_l$ are independent. We define the following covariance matrix,

$$\Sigma_\Delta = \begin{bmatrix} k_r|\Delta\phi_r| & 0 \\ 0 & k_l|\Delta\phi_l| \end{bmatrix}$$

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

To compute R_t we will assume that the errors in $\Delta\phi_r$ and $\Delta\phi_l$ are independent. We define the following covariance matrix,

$$\Sigma_\Delta = \begin{bmatrix} k_r|\Delta\phi_r| & 0 \\ 0 & k_l|\Delta\phi_l| \end{bmatrix}$$

The larger the movement, the larger our uncertainty

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

To compute R_t we will assume that the errors in $\Delta\phi_r$ and $\Delta\phi_l$ are independent. We define the following covariance matrix,

$$\Sigma_\Delta = \left[\begin{array}{cc} k_r|\Delta\phi_r| & 0 \\ 0 & k_l|\Delta\phi_l| \end{array} \right]$$

The larger the movement, the larger our uncertainty. Yet, this matrix is not R_t

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

To compute R_t we will assume that the errors in $\Delta\phi_r$ and $\Delta\phi_l$ are independent. We define the following covariance matrix,

$$\Sigma_\Delta = \begin{bmatrix} k_r|\Delta\phi_r| & 0 \\ 0 & k_l|\Delta\phi_l| \end{bmatrix}$$

The larger the movement, the larger our uncertainty. Yet, this matrix is not R_t. It describes our uncertainty in **control space** not in **state space**

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

To compute R_t we will assume that the errors in $\Delta\phi_r$ and $\Delta\phi_l$ are independent. We define the following covariance matrix,

$$\Sigma_\Delta = \begin{bmatrix} k_r|\Delta\phi_r| & 0 \\ 0 & k_l|\Delta\phi_l| \end{bmatrix}$$

The larger the movement, the larger our uncertainty. Yet, this matrix is not R_t. It describes our uncertainty in **control space** not in **state space**.

The control vector $[\Delta\phi_r, \Delta\phi_l]^T$ passes through the function g

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

To compute R_t we will assume that the errors in $\Delta\phi_r$ and $\Delta\phi_l$ are independent. We define the following covariance matrix,

$$\Sigma_\Delta = \begin{bmatrix} k_r|\Delta\phi_r| & 0 \\ 0 & k_l|\Delta\phi_l| \end{bmatrix}$$

The larger the movement, the larger our uncertainty. Yet, this matrix is not R_t. It describes our uncertainty in **control space** not in **state space**.

The control vector $[\Delta\phi_r, \Delta\phi_l]^T$ passes through the function g. Therefore, the covariance matrix R_t is formed by propagating Σ_Δ through g

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

To compute R_t we will assume that the errors in $\Delta\phi_r$ and $\Delta\phi_l$ are independent. We define the following covariance matrix,

$$\Sigma_\Delta = \begin{bmatrix} k_r|\Delta\phi_r| & 0 \\ 0 & k_l|\Delta\phi_l| \end{bmatrix}$$

The larger the movement, the larger our uncertainty. Yet, this matrix is not R_t. It describes our uncertainty in **control space** not in **state space**.

The control vector $[\Delta\phi_r, \Delta\phi_l]^T$ passes through the function g. Therefore, the covariance matrix R_t is formed by propagating Σ_Δ through g. To evaluate this uncertainty we determine the Jacobian of g with respect to the control vector $\begin{bmatrix}\Delta\phi_r & \Delta\phi_l\end{bmatrix}$

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

To compute R_t we will assume that the errors in $\Delta\phi_r$ and $\Delta\phi_l$ are independent. We define the following covariance matrix,

$$\Sigma_\Delta = \begin{bmatrix} k_r|\Delta\phi_r| & 0 \\ 0 & k_l|\Delta\phi_l| \end{bmatrix}$$

The larger the movement, the larger our uncertainty. Yet, this matrix is not R_t. It describes our uncertainty in **control space** not in **state space**.

The control vector $[\Delta\phi_r, \Delta\phi_l]^T$ passes through the function g. Therefore, the covariance matrix R_t is formed by propagating Σ_Δ through g. To evaluate this uncertainty we determine the Jacobian of g with respect to the control vector $\begin{bmatrix}\Delta\phi_r & \Delta\phi_l\end{bmatrix}$. We will call this Jacobian, J_t,

Our next challenge will be to compute R_t, which models the uncertainty introduced by the control signals: $\Delta\phi_r$ and $\Delta\phi_l$.

To compute R_t we will assume that the errors in $\Delta\phi_r$ and $\Delta\phi_l$ are independent. We define the following covariance matrix,

$$\Sigma_\Delta = \begin{bmatrix} k_r|\Delta\phi_r| & 0 \\ 0 & k_l|\Delta\phi_l| \end{bmatrix}$$

The larger the movement, the larger our uncertainty. Yet, this matrix is not R_t. It describes our uncertainty in **control space** not in **state space**.

The control vector $[\Delta\phi_r, \Delta\phi_l]^T$ passes through the function g. Therefore, the covariance matrix R_t is formed by propagating Σ_Δ through g. To evaluate this uncertainty we determine the Jacobian of g with respect to the control vector $\begin{bmatrix}\Delta\phi_r & \Delta\phi_l\end{bmatrix}$. We will call this Jacobian, J_t,

$$R_t = J_t \Sigma_\Delta J_t^T$$

$$R_t = J_t \Sigma_\Delta J_t^T$$

$$R_t = J_t \Sigma_\Delta J_t^T$$

Like G_t, J_t will be evaluated at the most likely value of x_{t-1}: μ_{t-1}.

$$R_t = J_t \Sigma_\Delta J_t^T$$

Like G_t, J_t will be evaluated at the most likely value of x_{t-1}: μ_{t-1}.

$$J_t \;=\; \begin{bmatrix} \frac{\partial g}{\partial \Delta\phi_r} & \frac{\partial g}{\partial \Delta\phi_l} \end{bmatrix}$$

$$R_t = J_t \Sigma_\Delta J_t^T$$

Like G_t, J_t will be evaluated at the most likely value of x_{t-1}: μ_{t-1}.

$$\begin{aligned} J_t &= \left[\frac{\partial g}{\partial \Delta\phi_r} \ \frac{\partial g}{\partial \Delta\phi_l}\right] \\ &= \frac{r}{2}\left[\begin{array}{cc} \cos(\theta+\Delta\theta) - f\sin(\theta+\Delta\theta) & \cos(\theta+\Delta\theta) + f\sin(\theta+\Delta\theta) \\ & \\ & \end{array}\right. \end{aligned}$$

$$R_t = J_t \Sigma_\Delta J_t^T$$

Like G_t, J_t will be evaluated at the most likely value of x_{t-1}: μ_{t-1}.

$$\begin{aligned} J_t &= \left[\frac{\partial g}{\partial \Delta \phi_r} \; \frac{\partial g}{\partial \Delta \phi_l} \right] \\ &= \frac{r}{2} \left[\begin{array}{cc} \cos(\theta + \Delta\theta) - f \sin(\theta + \Delta\theta) & \cos(\theta + \Delta\theta) + f \sin(\theta + \Delta\theta) \\ \sin(\theta + \Delta\theta) + f \cos(\theta + \Delta\theta) & \sin(\theta + \Delta\theta) - f \cos(\theta + \Delta\theta) \\ & \end{array} \right. \end{aligned}$$

$$R_t = J_t \Sigma_\Delta J_t^T$$

Like G_t, J_t will be evaluated at the most likely value of x_{t-1}: μ_{t-1}.

$$\begin{aligned} J_t &= \left[\frac{\partial g}{\partial \Delta\phi_r} \; \frac{\partial g}{\partial \Delta\phi_l}\right] \\ &= \frac{r}{2} \begin{bmatrix} \cos(\theta + \Delta\theta) - f \sin(\theta + \Delta\theta) & \cos(\theta + \Delta\theta) + f \sin(\theta + \Delta\theta) \\ \sin(\theta + \Delta\theta) + f \cos(\theta + \Delta\theta) & \sin(\theta + \Delta\theta) - f \cos(\theta + \Delta\theta) \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \end{aligned}$$

$$R_t = J_t \Sigma_\Delta J_t^T$$

Like G_t, J_t will be evaluated at the most likely value of x_{t-1}: μ_{t-1}.

$$\begin{aligned}
J_t &= \left[\frac{\partial g}{\partial \Delta\phi_r} \; \frac{\partial g}{\partial \Delta\phi_l} \right] \\
&= \frac{r}{2} \begin{bmatrix} \cos(\theta + \Delta\theta) - f\sin(\theta + \Delta\theta) & \cos(\theta + \Delta\theta) + f\sin(\theta + \Delta\theta) \\ \sin(\theta + \Delta\theta) + f\cos(\theta + \Delta\theta) & \sin(\theta + \Delta\theta) - f\cos(\theta + \Delta\theta) \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix}
\end{aligned}$$

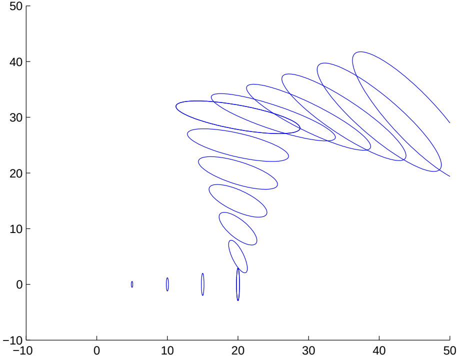
where $f = \frac{\Delta s}{l}$.

The plot below shows how the uncertainty grows for this example

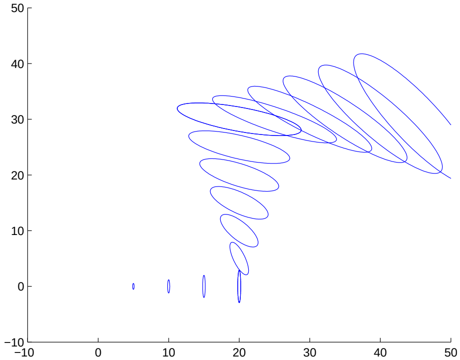
The plot below shows how the uncertainty grows for this example. The robot begins at $[0, 0, 0]^T$ and then moves to the right

The plot below shows how the uncertainty grows for this example. The robot begins at $[0, 0, 0]^T$ and then moves to the right. It then turns by 90°; continues straight; turns by -90°; then continues straight again.

The plot below shows how the uncertainty grows for this example. The robot begins at $[0, 0, 0]^T$ and then moves to the right. It then turns by 90°; continues straight; turns by -90°; then continues straight again.



The plot below shows how the uncertainty grows for this example. The robot begins at $[0, 0, 0]^T$ and then moves to the right. It then turns by 90°; continues straight; turns by -90°; then continues straight again.



The ellipses shown above are 50% confidence ellipses.

Example: Incorporating Point Features

We now extend the previous example by incorporating sensors

Example: Incorporating Point Features

We now extend the previous example by incorporating sensors. We assume that the sensor observation z_t is processed to yield a set of point features,

Example: Incorporating Point Features

We now extend the previous example by incorporating sensors. We assume that the sensor observation z_t is processed to yield a set of point features,

$$z_t = f(z_t^{raw}) \quad = \quad \{f_t^1, f_t^2, \ldots\}$$

Example: Incorporating Point Features

We now extend the previous example by incorporating sensors. We assume that the sensor observation z_t is processed to yield a set of point features,

$$
\begin{aligned}
z_t = f(z_t^{raw}) &= \{f_t^1, f_t^2, \ldots\} \\
&= \left\{ \begin{bmatrix} r_t^1 \\ \phi_t^1 \end{bmatrix}, \begin{bmatrix} r_t^2 \\ \phi_t^2 \end{bmatrix} \ldots \right\}
\end{aligned}
$$

Example: Incorporating Point Features

We now extend the previous example by incorporating sensors. We assume that the sensor observation z_t is processed to yield a set of point features,

$$\begin{aligned} z_t = f(z_t^{raw}) &= \{f_t^1, f_t^2, \ldots\} \\ &= \left\{ \begin{bmatrix} r_t^1 \\ \phi_t^1 \end{bmatrix}, \begin{bmatrix} r_t^2 \\ \phi_t^2 \end{bmatrix} \cdots \right\} \end{aligned}$$

where r_t^k and ϕ_t^k are the range and bearing of the k^{th} feature.

Example: Incorporating Point Features

We now extend the previous example by incorporating sensors. We assume that the sensor observation z_t is processed to yield a set of point features,

$$\begin{aligned} z_t = f(z_t^{raw}) &= \{f_t^1, f_t^2, \ldots\} \\ &= \left\{ \begin{bmatrix} r_t^1 \\ \phi_t^1 \end{bmatrix}, \begin{bmatrix} r_t^2 \\ \phi_t^2 \end{bmatrix} \ldots \right\} \end{aligned}$$

where r_t^k and ϕ_t^k are the range and bearing of the k^{th} feature.

We assume that these features are independent,

Example: Incorporating Point Features

We now extend the previous example by incorporating sensors. We assume that the sensor observation z_t is processed to yield a set of point features,

$$\begin{aligned} z_t = f(z_t^{raw}) &= \{f_t^1, f_t^2, \ldots\} \\ &= \left\{ \begin{bmatrix} r_t^1 \\ \phi_t^1 \end{bmatrix}, \begin{bmatrix} r_t^2 \\ \phi_t^2 \end{bmatrix} \ldots \right\} \end{aligned}$$

where r_t^k and ϕ_t^k are the range and bearing of the k^{th} feature.

We assume that these features are independent,

$$p(z_t|x_t, m) = \prod_i p(r_t^i, \phi_t^i|x_t, m)$$

Example: Incorporating Point Features

We now extend the previous example by incorporating sensors. We assume that the sensor observation z_t is processed to yield a set of point features,

$$\begin{aligned} z_t = f(z_t^{raw}) &= \{f_t^1, f_t^2, \ldots\} \\ &= \left\{ \begin{bmatrix} r_t^1 \\ \phi_t^1 \end{bmatrix}, \begin{bmatrix} r_t^2 \\ \phi_t^2 \end{bmatrix} \ldots \right\} \end{aligned}$$

where r_t^k and ϕ_t^k are the range and bearing of the k^{th} feature.

We assume that these features are independent,

$$p(z_t|x_t, m) = \prod_i p(r_t^i, \phi_t^i|x_t, m)$$

Therefore we can break the measurement update step into a sequence of steps

Example: Incorporating Point Features

We now extend the previous example by incorporating sensors. We assume that the sensor observation z_t is processed to yield a set of point features,

$$\begin{aligned} z_t = f(z_t^{raw}) &= \{f_t^1, f_t^2, \ldots\} \\ &= \left\{ \begin{bmatrix} r_t^1 \\ \phi_t^1 \end{bmatrix}, \begin{bmatrix} r_t^2 \\ \phi_t^2 \end{bmatrix} \cdots \right\} \end{aligned}$$

where r_t^k and ϕ_t^k are the range and bearing of the k^{th} feature.

We assume that these features are independent,

$$p(z_t|x_t, m) = \prod_i p(r_t^i, \phi_t^i | x_t, m)$$

Therefore we can break the measurement update step into a sequence of steps. In each step we update the belief for one feature.

The map m must be a *feature-based* map consisting of a list of features,

The map m must be a *feature-based* map consisting of a list of features,

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

The map m must be a *feature-based* map consisting of a list of features,

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

Assumption: We assume that all of the observed features can be correctly corresponded with features in the map.

The map m must be a *feature-based* map consisting of a list of features,

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

Assumption: We assume that all of the observed features can be correctly corresponded with features in the map.

The following function defines the **measurement model**

The map m must be a *feature-based* map consisting of a list of features,

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

Assumption: We assume that all of the observed features can be correctly corresponded with features in the map.

The following function defines the **measurement model**. It is formulated such that the ith feature observed at time t corresponds to the jth feature in the map (not all map features will be perceived at each time step)

The map m must be a *feature-based* map consisting of a list of features,

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

Assumption: We assume that all of the observed features can be correctly corresponded with features in the map.

The following function defines the **measurement model**. It is formulated such that the ith feature observed at time t corresponds to the jth feature in the map (not all map features will be perceived at each time step).

$$z_t^i = \begin{bmatrix} r_t^i \\ \phi_t^i \end{bmatrix} = \begin{bmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ atan2(m_{j,y} - y, m_{j,x} - x) - \theta \end{bmatrix} + \begin{bmatrix} \epsilon_r \\ \epsilon_\phi \end{bmatrix}$$

The map m must be a *feature-based* map consisting of a list of features,

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

Assumption: We assume that all of the observed features can be correctly corresponded with features in the map.

The following function defines the **measurement model**. It is formulated such that the ith feature observed at time t corresponds to the jth feature in the map (not all map features will be perceived at each time step).

$$\begin{aligned} z_t^i = \begin{bmatrix} r_t^i \\ \phi_t^i \end{bmatrix} &= \begin{bmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ atan2(m_{j,y} - y, m_{j,x} - x) - \theta \end{bmatrix} + \begin{bmatrix} \epsilon_r \\ \epsilon_\phi \end{bmatrix} \\ &= h(x_t) \end{aligned}$$

The map m must be a *feature-based* map consisting of a list of features,

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

Assumption: We assume that all of the observed features can be correctly corresponded with features in the map.

The following function defines the **measurement model**. It is formulated such that the ith feature observed at time t corresponds to the jth feature in the map (not all map features will be perceived at each time step).

$$\begin{aligned} z_t^i = \begin{bmatrix} r_t^i \\ \phi_t^i \end{bmatrix} &= \begin{bmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ atan2(m_{j,y} - y, m_{j,x} - x) - \theta \end{bmatrix} + \begin{bmatrix} \epsilon_r \\ \epsilon_\phi \end{bmatrix} \\ &= h(x_t) \end{aligned}$$

where the ϵ terms represent zero-mean Gaussian error variables with standard deviations σ_r and σ_ϕ.

Consider the EKF measurement update:

Consider the EKF measurement update:

4. $$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$$

Consider the EKF measurement update:

4. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$

5. $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

Consider the EKF measurement update:

4. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$

5. $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6. $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

Consider the EKF measurement update:

4. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$

5. $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6. $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

We incorporate point features by replacing these steps with a loop over observed features

Consider the EKF measurement update:

4. $\quad K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$

5. $\quad \mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6. $\quad \Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

We incorporate point features by replacing these steps with a loop over observed features. Within this loop we predict the features that we should observe, which are given by the measurement model applied to $\bar{\mu}_t$,

Consider the EKF measurement update:

4. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$

5. $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6. $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

We incorporate point features by replacing these steps with a loop over observed features. Within this loop we predict the features that we should observe, which are given by the measurement model applied to $\bar{\mu}_t$,

$$\hat{z}_t^i = \begin{bmatrix} \sqrt{(m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2} \\ atan2(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \end{bmatrix}$$

Consider the EKF measurement update:

4. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$

5. $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6. $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

We incorporate point features by replacing these steps with a loop over observed features. Within this loop we predict the features that we should observe, which are given by the measurement model applied to $\bar{\mu}_t$,

$$\begin{aligned}\hat{z}_t^i &= \begin{bmatrix} \sqrt{(m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2} \\ atan2(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \end{bmatrix} \\ &= h(\bar{\mu}_t)\end{aligned}$$

Consider the EKF measurement update:

4. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$

5. $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6. $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

We incorporate point features by replacing these steps with a loop over observed features. Within this loop we predict the features that we should observe, which are given by the measurement model applied to $\bar{\mu}_t$,

$$\begin{aligned}\hat{z}_t^i &= \begin{bmatrix} \sqrt{(m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2} \\ atan2(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \end{bmatrix} \\ &= h(\bar{\mu}_t)\end{aligned}$$

The measurement update (steps 4-6) of the standard EKF algorithm is now replaced with the following loop:

for all observed features $z_t^i = (r_t^i, \phi_t^i)$

$j =$ The map index of feature i

The measurement update (steps 4-6) of the standard EKF algorithm is now replaced with the following loop:

for all observed features $z_t^i = (r_t^i, \phi_t^i)$

$j =$ The map index of feature i

$$H_t^i = \begin{bmatrix} -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y} - \bar{\mu}_{t,y}}{q} & -\frac{m_{j,x} - \bar{\mu}_{t,x}}{q} & -1 \end{bmatrix}$$

The measurement update (steps 4-6) of the standard EKF algorithm is now replaced with the following loop:

for all observed features $z_t^i = (r_t^i, \phi_t^i)$

$j =$ The map index of feature i

$$H_t^i = \begin{bmatrix} -\frac{m_{j,x}-\bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y}-\bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y}-\bar{\mu}_{t,y}}{q} & -\frac{m_{j,x}-\bar{\mu}_{t,x}}{q} & -1 \end{bmatrix}$$

where $q = (x - m_{j,x})^2 + (y - m_{j,y})^2$

The measurement update (steps 4-6) of the standard EKF algorithm is now replaced with the following loop:

for all observed features $z_t^i = (r_t^i, \phi_t^i)$

$j =$ The map index of feature i

$$H_t^i = \begin{bmatrix} -\frac{m_{j,x}-\bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y}-\bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y}-\bar{\mu}_{t,y}}{q} & -\frac{m_{j,x}-\bar{\mu}_{t,x}}{q} & -1 \end{bmatrix}$$

where $q = (x - m_{j,x})^2 + (y - m_{j,y})^2$

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$$

The measurement update (steps 4-6) of the standard EKF algorithm is now replaced with the following loop:

for all observed features $z_t^i = (r_t^i, \phi_t^i)$

$j =$ The map index of feature i

$$H_t^i = \begin{bmatrix} -\frac{m_{j,x}-\bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y}-\bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y}-\bar{\mu}_{t,y}}{q} & -\frac{m_{j,x}-\bar{\mu}_{t,x}}{q} & -1 \end{bmatrix}$$

where $q = (x - m_{j,x})^2 + (y - m_{j,y})^2$

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$$

$$\bar{\mu}_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$$

The measurement update (steps 4-6) of the standard EKF algorithm is now replaced with the following loop:

for all observed features $z_t^i = (r_t^i, \phi_t^i)$

$j =$ The map index of feature i

$$H_t^i = \begin{bmatrix} -\frac{m_{j,x}-\bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y}-\bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y}-\bar{\mu}_{t,y}}{q} & -\frac{m_{j,x}-\bar{\mu}_{t,x}}{q} & -1 \end{bmatrix}$$

where $q = (x - m_{j,x})^2 + (y - m_{j,y})^2$

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$$

$$\bar{\mu}_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$$

$$\bar{\Sigma}_t = (I - K_t H_t)\bar{\Sigma}_t$$

The measurement update (steps 4-6) of the standard EKF algorithm is now replaced with the following loop:

for all observed features $z_t^i = (r_t^i, \phi_t^i)$

$j =$ The map index of feature i

$$H_t^i = \begin{bmatrix} -\frac{m_{j,x}-\bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y}-\bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y}-\bar{\mu}_{t,y}}{q} & -\frac{m_{j,x}-\bar{\mu}_{t,x}}{q} & -1 \end{bmatrix}$$

where $q = (x - m_{j,x})^2 + (y - m_{j,y})^2$

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$$

$$\bar{\mu}_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$$

$$\bar{\Sigma}_t = (I - K_t H_t)\bar{\Sigma}_t$$

end for

The measurement update (steps 4-6) of the standard EKF algorithm is now replaced with the following loop:

for all observed features $z_t^i = (r_t^i, \phi_t^i)$

$j =$ The map index of feature i

$$H_t^i = \begin{bmatrix} -\frac{m_{j,x}-\bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y}-\bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y}-\bar{\mu}_{t,y}}{q} & -\frac{m_{j,x}-\bar{\mu}_{t,x}}{q} & -1 \end{bmatrix}$$

where $q = (x - m_{j,x})^2 + (y - m_{j,y})^2$

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$$

$$\bar{\mu}_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$$

$$\bar{\Sigma}_t = (I - K_t H_t)\bar{\Sigma}_t$$

end for

$$\mu_t = \bar{\mu}_t$$

The measurement update (steps 4-6) of the standard EKF algorithm is now replaced with the following loop:

for all observed features $z_t^i = (r_t^i, \phi_t^i)$

$j =$ The map index of feature i

$$H_t^i = \begin{bmatrix} -\frac{m_{j,x}-\bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y}-\bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y}-\bar{\mu}_{t,y}}{q} & -\frac{m_{j,x}-\bar{\mu}_{t,x}}{q} & -1 \end{bmatrix}$$

where $q = (x - m_{j,x})^2 + (y - m_{j,y})^2$

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$$

$$\bar{\mu}_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$$

$$\bar{\Sigma}_t = (I - K_t H_t)\bar{\Sigma}_t$$

end for

$$\mu_t = \bar{\mu}_t$$

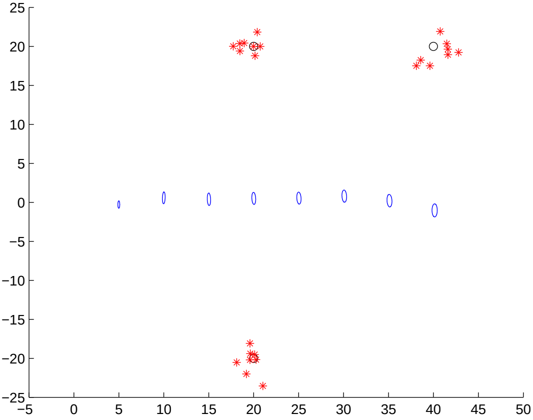
$$\Sigma_t = \bar{\Sigma}_t$$

The plot below shows the belief state of a robot moving to the right

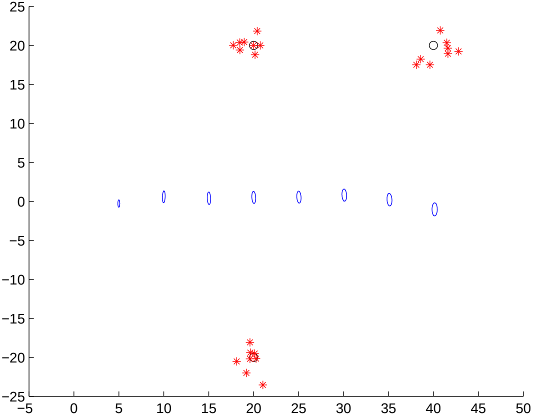
The plot below shows the belief state of a robot moving to the right. The three black circles are landmarks

The plot below shows the belief state of a robot moving to the right. The three black circles are landmarks. The red stars surrounding each landmark are the perceived positions of the landmark over time

The plot below shows the belief state of a robot moving to the right. The three black circles are landmarks. The red stars surrounding each landmark are the perceived positions of the landmark over time.



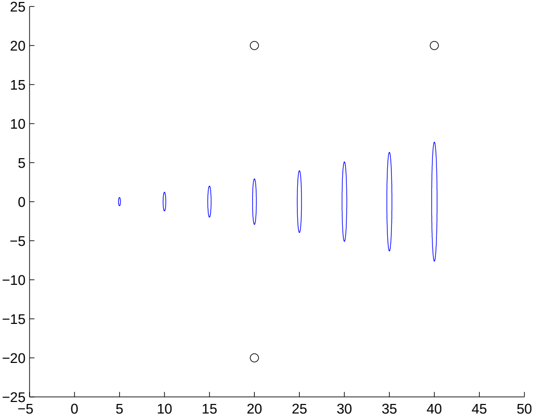
The plot below shows the belief state of a robot moving to the right. The three black circles are landmarks. The red stars surrounding each landmark are the perceived positions of the landmark over time.



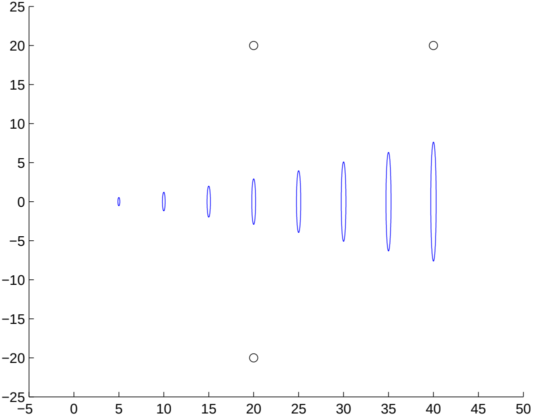
The uncertainty ellipses remain bounded in size.

Without incorporating the measurement update we would have the following situation,

Without incorporating the measurement update we would have the following situation,



Without incorporating the measurement update we would have the following situation,



Here the uncertainty ellipses continue to grow over time.