Localization: Part 6
The Kalman Filter

Computer Science 4766/6912

Department of Computer Science
Memorial University of Newfoundland

July 11, 2018

Introduction

- The Kalman filter is perhaps the most widely-known implementation of Bayes filter

Introduction

- The Kalman filter is perhaps the most widely-known implementation of Bayes filter
- Numerous applications outside of robot localization

Introduction

- The Kalman filter is perhaps the most widely-known implementation of Bayes filter
- Numerous applications outside of robot localization
- Based upon a Gaussian belief representation

Introduction

- The Kalman filter is perhaps the most widely-known implementation of Bayes filter
- Numerous applications outside of robot localization
- Based upon a Gaussian belief representation
  - Belief represented as the mean μ_t and covariance matrix Σ_t of the Guassian

Introduction

- The Kalman filter is perhaps the most widely-known implementation of Bayes filter
- Numerous applications outside of robot localization
- Based upon a Gaussian belief representation
  - Belief represented as the mean μ_t and covariance matrix Σ_t of the Guassian
- Assumes that the system update equations are **linear**

Introduction

- The Kalman filter is perhaps the most widely-known implementation of Bayes filter
- Numerous applications outside of robot localization
- Based upon a Gaussian belief representation
  - Belief represented as the mean μ_t and covariance matrix Σ_t of the Guassian
- Assumes that the system update equations are **linear**
  - Non-linearity handled by *linearization*: **The Extended Kalman Filter**

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- A_t is a matrix of size

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- A_t is a matrix of size

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- A_t is a matrix of size $n \times n$

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- A_t is a matrix of size $n \times n$
- B_t is a matrix of size

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- A_t is a matrix of size $n \times n$
- B_t is a matrix of size

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- A_t is a matrix of size $n \times n$
- B_t is a matrix of size $n \times m$

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- A_t is a matrix of size $n \times n$
- B_t is a matrix of size $n \times m$
- ϵ_t is a Gaussian random vector of size

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- A_t is a matrix of size $n \times n$
- B_t is a matrix of size $n \times m$
- ϵ_t is a Gaussian random vector of size

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- A_t is a matrix of size $n \times n$
- B_t is a matrix of size $n \times m$
- ϵ_t is a Gaussian random vector of size n

Assumptions

- "The system" is some process evolving over time, through control inputs u_t and measurements z_t
- It has an unknown state x_t and unknown past state x_{t-1}
- x_t, u_t, and z_t are all column vectors with n, m, and k elements, respectively
- We assume that x_t evolves through the following **linear** update equation,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- A_t is a matrix of size $n \times n$
- B_t is a matrix of size $n \times m$
- ϵ_t is a Gaussian random vector of size n
    - Zero mean and covariance matrix R_t

- The measurements z_t are assumed to be another **linear** function of the system state,

- The measurements z_t are assumed to be another **linear** function of the system state,

- The measurements z_t are assumed to be another **linear** function of the system state,

$$z_t = C_t x_t + \delta_t$$

- The measurements z_t are assumed to be another **linear** function of the system state,

$$z_t = C_t x_t + \delta_t$$

- C_t is a matrix of size

- The measurements z_t are assumed to be another **linear** function of the system state,

$$z_t = C_t x_t + \delta_t$$

- C_t is a matrix of size

- The measurements z_t are assumed to be another **linear** function of the system state,

$$z_t = C_t x_t + \delta_t$$

- C_t is a matrix of size $k \times n$

- The measurements z_t are assumed to be another **linear** function of the system state,

$$z_t = C_t x_t + \delta_t$$

- C_t is a matrix of size $k \times n$
- δ_t is a Gaussian random vector of size k

- The measurements z_t are assumed to be another **linear** function of the system state,

$$z_t = C_t x_t + \delta_t$$

- C_t is a matrix of size $k \times n$
- δ_t is a Gaussian random vector of size k
  - Zero mean and covariance matrix Q_t

- The measurements z_t are assumed to be another **linear** function of the system state,

$$z_t = C_t x_t + \delta_t$$

- C_t is a matrix of size $k \times n$
- δ_t is a Gaussian random vector of size k
  - Zero mean and covariance matrix Q_t
- It is necessary for the noise vectors $(\epsilon_0, \epsilon_1, \ldots \epsilon_t, \delta_0, \delta_1, \ldots, \delta_t)$ to be mutually independent.

- The measurements z_t are assumed to be another **linear** function of the system state,

$$z_t = C_t x_t + \delta_t$$

- C_t is a matrix of size $k \times n$
- δ_t is a Gaussian random vector of size k
  - Zero mean and covariance matrix Q_t
- It is necessary for the noise vectors (ϵ_0, ϵ_1, $\ldots \epsilon_t$, δ_0, δ_1, $\ldots$, δ_t) to be mutually independent.

- The measurements z_t are assumed to be another **linear** function of the system state,

$$z_t = C_t x_t + \delta_t$$

- C_t is a matrix of size $k \times n$
- δ_t is a Gaussian random vector of size k
  - Zero mean and covariance matrix Q_t
- It is necessary for the noise vectors $(\epsilon_0, \epsilon_1, \ldots \epsilon_t, \delta_0, \delta_1, \ldots, \delta_t)$ to be mutually independent.

$bel(x_t)$ is represented by mean μ_t and covariance matrix Σ_t

- The measurements z_t are assumed to be another **linear** function of the system state,

$$z_t = C_t x_t + \delta_t$$

- C_t is a matrix of size $k \times n$
- δ_t is a Gaussian random vector of size k
    - Zero mean and covariance matrix Q_t
- It is necessary for the noise vectors (ϵ_0, ϵ_1, $\ldots \epsilon_t$, δ_0, δ_1, $\ldots$, δ_t) to be mutually independent.

$bel(x_t)$ is represented by mean μ_t and covariance matrix Σ_t

- The initial belief $bel(x_0)$ must have a Gaussian p.d.f with mean μ_0 and covariance matrix Σ_0

- The measurements z_t are assumed to be another **linear** function of the system state,

$$z_t = C_t x_t + \delta_t$$

- C_t is a matrix of size $k \times n$
- δ_t is a Gaussian random vector of size k
  - Zero mean and covariance matrix Q_t
- It is necessary for the noise vectors $(\epsilon_0, \epsilon_1, \ldots \epsilon_t, \delta_0, \delta_1, \ldots, \delta_t)$ to be mutually independent.

$bel(x_t)$ is represented by mean μ_t and covariance matrix Σ_t

- The initial belief $bel(x_0)$ must have a Gaussian p.d.f with mean μ_0 and covariance matrix Σ_0
- If the above assumptions hold, the belief $bel(x_t)$ will always be Gaussian

The Kalman Filter Algorithm

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

The Kalman Filter Algorithm

1. Kalman_Filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2. $\quad \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$

The Kalman Filter Algorithm

1. Kalman_Filter$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2. $\quad \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\quad \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

The Kalman Filter Algorithm

1. Kalman_Filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2. $\quad \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\quad \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
4. $\quad K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$

The Kalman Filter Algorithm

1. Kalman_Filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
4. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t)$

The Kalman Filter Algorithm

1. Kalman_Filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
4. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t)$
6. $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$

The Kalman Filter Algorithm

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

The Kalman Filter Algorithm

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

Lines 2 and 3 implement the prediction step, which determines $\overline{bel}(x_t)$.

The Kalman Filter Algorithm

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

Lines 2 and 3 implement the prediction step, which determines $\overline{bel}(x_t)$.

Lines 4 - 6 implement the measurement update, which incorporates z_t to determine $bel(x_t)$

The Kalman Filter Algorithm

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

Lines 2 and 3 implement the prediction step, which determines $\overline{bel}(x_t)$.

Lines 4 - 6 implement the measurement update, which incorporates z_t to determine $bel(x_t)$. On line 4 the *Kalman gain*, K_t, is computed

The Kalman Filter Algorithm

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

Lines 2 and 3 implement the prediction step, which determines $\overline{bel}(x_t)$.

Lines 4 - 6 implement the measurement update, which incorporates z_t to determine $bel(x_t)$. On line 4 the *Kalman gain*, K_t, is computed. K_t specifies the degree of trust placed in z_t over $\overline{bel}(x_t)$.

On line 5 the actual measurement z_t is compared with the **measurement prediction** $C_t \bar{\mu}_t$

The Kalman Filter Algorithm

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

Lines 2 and 3 implement the prediction step, which determines $\overline{bel}(x_t)$.

Lines 4 - 6 implement the measurement update, which incorporates z_t to determine $bel(x_t)$. On line 4 the *Kalman gain*, K_t, is computed. K_t specifies the degree of trust placed in z_t over $\overline{bel}(x_t)$.

On line 5 the actual measurement z_t is compared with the **measurement prediction** $C_t \bar{\mu}_t$. This difference is known as the **innovation**

The Kalman Filter Algorithm

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

Lines 2 and 3 implement the prediction step, which determines $\overline{bel}(x_t)$.

Lines 4 - 6 implement the measurement update, which incorporates z_t to determine $bel(x_t)$. On line 4 the *Kalman gain*, K_t, is computed. K_t specifies the degree of trust placed in z_t over $\overline{bel}(x_t)$.

On line 5 the actual measurement z_t is compared with the **measurement prediction** $C_t \bar{\mu}_t$. This difference is known as the **innovation**. It describes how different z_t is from what was expected.

Example: Simple 1-D Robot

We apply the Kalman filter to a simple robot living in a 1-D world

Example: Simple 1-D Robot

We apply the Kalman filter to a simple robot living in a 1-D world. The "state vector" is just the scalar position x

Example: Simple 1-D Robot

We apply the Kalman filter to a simple robot living in a 1-D world. The "state vector" is just the scalar position x. The matrices A_t, B_t, and C_t are assumed constant and 1-D—they reduce to scalars a, b, and c

Example: Simple 1-D Robot

We apply the Kalman filter to a simple robot living in a 1-D world. The "state vector" is just the scalar position x. The matrices A_t, B_t, and C_t are assumed constant and 1-D—they reduce to scalars a, b, and c. The covariance matrices R_t and Q_t are also reduced to scalars r and q

Example: Simple 1-D Robot

We apply the Kalman filter to a simple robot living in a 1-D world. The "state vector" is just the scalar position x. The matrices A_t, B_t, and C_t are assumed constant and 1-D—they reduce to scalars a, b, and c. The covariance matrices R_t and Q_t are also reduced to scalars r and q. (Here, $r = 0.25$ and $q = 1$)

Example: Simple 1-D Robot

We apply the Kalman filter to a simple robot living in a 1-D world. The "state vector" is just the scalar position x. The matrices A_t, B_t, and C_t are assumed constant and 1-D—they reduce to scalars a, b, and c. The covariance matrices R_t and Q_t are also reduced to scalars r and q. (Here, $r = 0.25$ and $q = 1$)

We assume that unless it is commanded to move, the robot will maintain its current position: $a = 1$

Example: Simple 1-D Robot

We apply the Kalman filter to a simple robot living in a 1-D world. The "state vector" is just the scalar position x. The matrices A_t, B_t, and C_t are assumed constant and 1-D—they reduce to scalars a, b, and c. The covariance matrices R_t and Q_t are also reduced to scalars r and q. (Here, $r = 0.25$ and $q = 1$)

We assume that unless it is commanded to move, the robot will maintain its current position: $a = 1$

The control signal u_t is the distance the robot is commanded to move: $b = 1$ (the robot obeys and moves the commanded distance)

Example: Simple 1-D Robot

We apply the Kalman filter to a simple robot living in a 1-D world. The "state vector" is just the scalar position x. The matrices A_t, B_t, and C_t are assumed constant and 1-D—they reduce to scalars a, b, and c. The covariance matrices R_t and Q_t are also reduced to scalars r and q. (Here, $r = 0.25$ and $q = 1$)

We assume that unless it is commanded to move, the robot will maintain its current position: $a = 1$

The control signal u_t is the distance the robot is commanded to move: $b = 1$ (the robot obeys and moves the commanded distance)

The robot has a position sensor which gives 1-D position directly: $c = 1$

Example: Simple 1-D Robot

We apply the Kalman filter to a simple robot living in a 1-D world. The "state vector" is just the scalar position x. The matrices A_t, B_t, and C_t are assumed constant and 1-D—they reduce to scalars a, b, and c. The covariance matrices R_t and Q_t are also reduced to scalars r and q. (Here, $r = 0.25$ and $q = 1$)

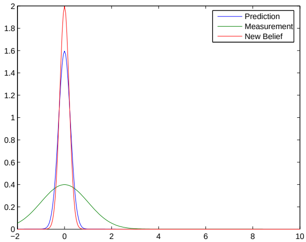
We assume that unless it is commanded to move, the robot will maintain its current position: $a = 1$

The control signal u_t is the distance the robot is commanded to move: $b = 1$ (the robot obeys and moves the commanded distance)

The robot has a position sensor which gives 1-D position directly: $c = 1$

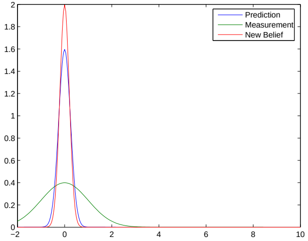
The robot begins with $\mu_0 = 0, \sigma_0 = 0$...

$x_t = 0 : u_t = 0, z = 0$

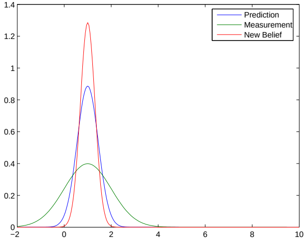


$x_t = 1 : u_t = 1, z = 1$

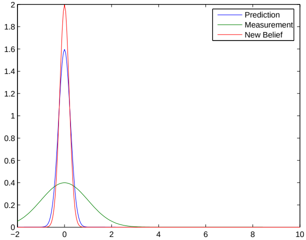
$$x_t = 0 : u_t = 0, z = 0$$



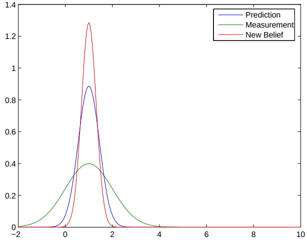
$$x_t = 1 : u_t = 1, z = 1$$



$x_t = 0 : u_t = 0, z = 0$



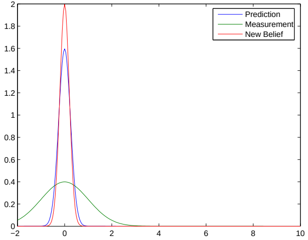
$x_t = 1 : u_t = 1, z = 1$



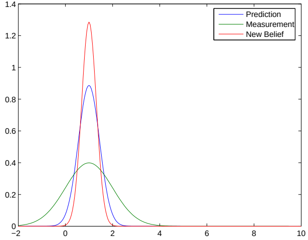
$x_t = 2 : u_t = 1,$ $z = 4(\text{error})$

$x_t = 0 : u_t = -2, z = 0$

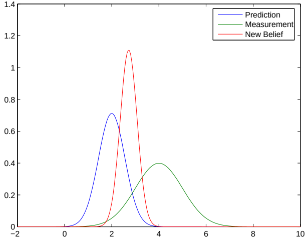
$x_t = 0 : u_t = 0, z = 0$



$x_t = 1 : u_t = 1, z = 1$

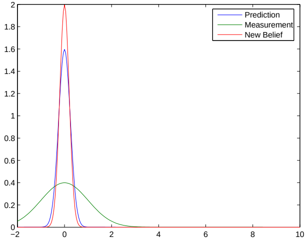


$x_t = 2 : u_t = 1, z = 4(\text{error})$

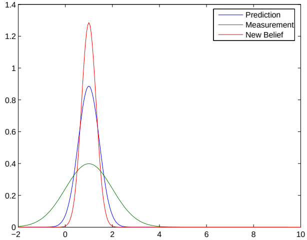


$x_t = 0 : u_t = -2, z = 0$

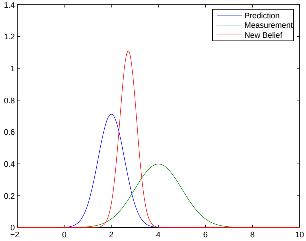
$x_t = 0 : u_t = 0, z = 0$



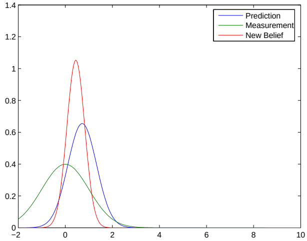
$x_t = 1 : u_t = 1, z = 1$



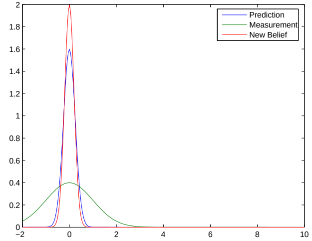
$x_t = 2 : u_t = 1,$ $z = 4(\text{error})$



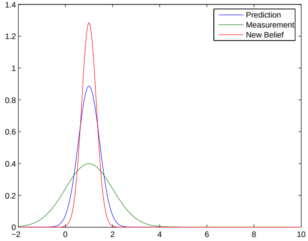
$x_t = 0 : u_t = -2, z = 0$



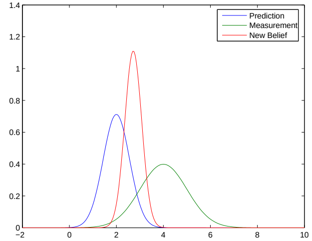
$x_t = 0 : u_t = 0, z = 0$



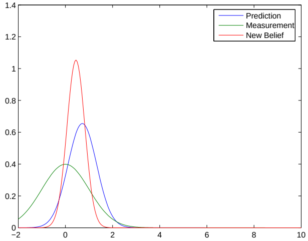
$x_t = 1 : u_t = 1, z = 1$



$x_t = 2 : u_t = 1, z = 4(\text{error})$

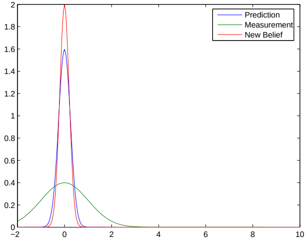


$x_t = 0 : u_t = -2, z = 0$

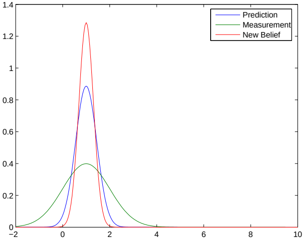


Notice when $x_t = 2$ we get an erroneous sensor value of $z = 4$

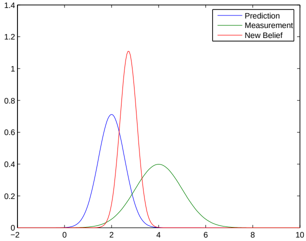
$x_t = 0 : u_t = 0, z = 0$



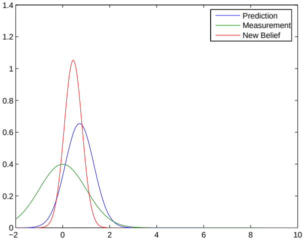
$x_t = 1 : u_t = 1, z = 1$



$x_t = 2 : u_t = 1, z = 4(\text{error})$



$x_t = 0 : u_t = -2, z = 0$



Notice when $x_t = 2$ we get an erroneous sensor value of $z = 4$. The new belief is closer to 2 than 4 because $\bar{\sigma} < q$

Example: Another 1-D Robot, but with 2-D State

Consider a robot constrained to move along a 1-D track

Example: Another 1-D Robot, but with 2-D State

Consider a robot constrained to move along a 1-D track. We wish to track the robot's position and speed over time

Example: Another 1-D Robot, but with 2-D State

Consider a robot constrained to move along a 1-D track. We wish to track the robot's position and speed over time. The state vector is,

Example: Another 1-D Robot, but with 2-D State

Consider a robot constrained to move along a 1-D track. We wish to track the robot's position and speed over time. The state vector is,

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

where p_t is position and v_t is velocity

Example: Another 1-D Robot, but with 2-D State

Consider a robot constrained to move along a 1-D track. We wish to track the robot's position and speed over time. The state vector is,

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

where p_t is position and v_t is velocity. The control input, u_t, is a *force* applied on the robot, which has a mass of m

Example: Another 1-D Robot, but with 2-D State

Consider a robot constrained to move along a 1-D track. We wish to track the robot's position and speed over time. The state vector is,

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

where p_t is position and v_t is velocity. The control input, u_t, is a *force* applied on the robot, which has a mass of m. From Newton's second law we know that $force = mass \times acceleration$. Thus,

Example: Another 1-D Robot, but with 2-D State

Consider a robot constrained to move along a 1-D track. We wish to track the robot's position and speed over time. The state vector is,

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

where p_t is position and v_t is velocity. The control input, u_t, is a *force* applied on the robot, which has a mass of m. From Newton's second law we know that $force = mass \times acceleration$. Thus,

$$u = m\frac{dv}{dt}$$

which means that the acceleration $\frac{dv}{dt} = \frac{u}{m}$

Example: Another 1-D Robot, but with 2-D State

Consider a robot constrained to move along a 1-D track. We wish to track the robot's position and speed over time. The state vector is,

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

where p_t is position and v_t is velocity. The control input, u_t, is a *force* applied on the robot, which has a mass of m. From Newton's second law we know that $force = mass \times acceleration$. Thus,

$$u = m\frac{dv}{dt}$$

which means that the acceleration $\frac{dv}{dt} = \frac{u}{m}$. We will assume that the time period between discrete updates, Δt, is sufficiently small such that,

Example: Another 1-D Robot, but with 2-D State

Consider a robot constrained to move along a 1-D track. We wish to track the robot's position and speed over time. The state vector is,

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

where p_t is position and v_t is velocity. The control input, u_t, is a *force* applied on the robot, which has a mass of m. From Newton's second law we know that *force* = *mass* × *acceleration*. Thus,

$$u = m \frac{dv}{dt}$$

which means that the acceleration $\frac{dv}{dt} = \frac{u}{m}$. We will assume that the time period between discrete updates, Δt, is sufficiently small such that,

$$\frac{dv}{dt} \approx \frac{v_t - v_{t-1}}{\Delta t}$$

Given these assumptions we can give update equations for our two main variables,

Given these assumptions we can give update equations for our two main variables,

$$p_t \quad = \quad p_{t-1} + \Delta t \, v_{t-1} + \mathrm{Noise}$$

Given these assumptions we can give update equations for our two main variables,

$$
\begin{aligned}
p_t &= p_{t-1} + \Delta t \, v_{t-1} + \text{Noise} \\
v_t &= v_{t-1} + \frac{\Delta t}{m} u_t + \text{Noise}
\end{aligned}
$$

Given these assumptions we can give update equations for our two main variables,

$$
\begin{aligned}
p_t &= p_{t-1} + \Delta t \, v_{t-1} + \text{Noise} \\
v_t &= v_{t-1} + \frac{\Delta t}{m} u_t + \text{Noise}
\end{aligned}
$$

In order to apply the Kalman filter, we must write these equations together in matrix form,

Given these assumptions we can give update equations for our two main variables,

$$
\begin{aligned}
p_t &= p_{t-1} + \Delta t \, v_{t-1} + \mathrm{Noise} \\
v_t &= v_{t-1} + \frac{\Delta t}{m} u_t + \mathrm{Noise}
\end{aligned}
$$

In order to apply the Kalman filter, we must write these equations together in matrix form,

$$
\begin{bmatrix} p_t \\ v_t \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{t-1} \\ v_{t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\Delta t}{m} \end{bmatrix} u_t + \epsilon_t
$$

Given these assumptions we can give update equations for our two main variables,

$$
\begin{aligned}
p_t &= p_{t-1} + \Delta t \, v_{t-1} + \mathrm{Noise} \\
v_t &= v_{t-1} + \frac{\Delta t}{m} u_t + \mathrm{Noise}
\end{aligned}
$$

In order to apply the Kalman filter, we must write these equations together in matrix form,

$$
\begin{bmatrix} p_t \\ v_t \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{t-1} \\ v_{t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\Delta t}{m} \end{bmatrix} u_t + \epsilon_t
$$

where ϵ_t represents additive Gaussian noise with covariance matrix R_t

Given these assumptions we can give update equations for our two main variables,

$$
\begin{aligned}
p_t &= p_{t-1} + \Delta t \, v_{t-1} + \text{Noise} \\
v_t &= v_{t-1} + \frac{\Delta t}{m} u_t + \text{Noise}
\end{aligned}
$$

In order to apply the Kalman filter, we must write these equations together in matrix form,

$$
\begin{bmatrix} p_t \\ v_t \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{t-1} \\ v_{t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\Delta t}{m} \end{bmatrix} u_t + \epsilon_t
$$

where ϵ_t represents additive Gaussian noise with covariance matrix R_t. This equation is now in the standard form required by the Kalman filter:

Given these assumptions we can give update equations for our two main variables,

$$\begin{aligned} p_t &= p_{t-1} + \Delta t \, v_{t-1} + \text{Noise} \\ v_t &= v_{t-1} + \frac{\Delta t}{m} u_t + \text{Noise} \end{aligned}$$

In order to apply the Kalman filter, we must write these equations together in matrix form,

$$\begin{bmatrix} p_t \\ v_t \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{t-1} \\ v_{t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\Delta t}{m} \end{bmatrix} u_t + \epsilon_t$$

where ϵ_t represents additive Gaussian noise with covariance matrix R_t. This equation is now in the standard form required by the Kalman filter:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

Given these assumptions we can give update equations for our two main variables,

$$\begin{aligned} p_t &= p_{t-1} + \Delta t\, v_{t-1} + \text{Noise} \\ v_t &= v_{t-1} + \frac{\Delta t}{m} u_t + \text{Noise} \end{aligned}$$

In order to apply the Kalman filter, we must write these equations together in matrix form,

$$\begin{bmatrix} p_t \\ v_t \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{t-1} \\ v_{t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\Delta t}{m} \end{bmatrix} u_t + \epsilon_t$$

where ϵ_t represents additive Gaussian noise with covariance matrix R_t. This equation is now in the standard form required by the Kalman filter:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

Assume that this robot is also equipped with a position sensor (subject to noise of course)...

The standard form for generating measurements is as follows,

$$z_t = C_t x_t + \delta_t$$

The standard form for generating measurements is as follows,

$$z_t = C_t x_t + \delta_t$$

In our case the matrix C_t is quite simple,

The standard form for generating measurements is as follows,

$$z_t = C_t x_t + \delta_t$$

In our case the matrix C_t is quite simple,

$$z_t = [1\ 0] \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \delta_t$$

The standard form for generating measurements is as follows,

$$z_t = C_t x_t + \delta_t$$

In our case the matrix C_t is quite simple,

$$z_t = [1\ 0] \left[\begin{array}{c} p_t \\ v_t \end{array} \right] + \delta_t$$

where δ_t represents Gaussian noise with covariance matrix Q_t

The standard form for generating measurements is as follows,

$$z_t = C_t x_t + \delta_t$$

In our case the matrix C_t is quite simple,

$$z_t = [1\ 0] \left[\begin{array}{c} p_t \\ v_t \end{array} \right] + \delta_t$$

where δ_t represents Gaussian noise with covariance matrix Q_t. In this case, Q_t is just a variance

The standard form for generating measurements is as follows,

$$z_t = C_t x_t + \delta_t$$

In our case the matrix C_t is quite simple,

$$z_t = [1\ 0] \left[\begin{array}{c} p_t \\ v_t \end{array} \right] + \delta_t$$

where δ_t represents Gaussian noise with covariance matrix Q_t. In this case, Q_t is just a variance.

The Kalman filter can now be applied

The standard form for generating measurements is as follows,

$$z_t = C_t x_t + \delta_t$$

In our case the matrix C_t is quite simple,

$$z_t = [1\ 0] \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \delta_t$$

where δ_t represents Gaussian noise with covariance matrix Q_t. In this case, Q_t is just a variance.

The Kalman filter can now be applied. The parameters we will use are as follows:

The standard form for generating measurements is as follows,

$$z_t = C_t x_t + \delta_t$$

In our case the matrix C_t is quite simple,

$$z_t = [1\ 0] \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \delta_t$$

where δ_t represents Gaussian noise with covariance matrix Q_t. In this case, Q_t is just a variance.

The Kalman filter can now be applied. The parameters we will use are as follows:

- Covariance matrix of motion equation: $R = \begin{bmatrix} 0.2 & 0.05 \\ 0.05 & 0.1 \end{bmatrix}$

The standard form for generating measurements is as follows,

$$z_t = C_t x_t + \delta_t$$

In our case the matrix C_t is quite simple,

$$z_t = [1 \ 0] \left[\begin{array}{c} p_t \\ v_t \end{array} \right] + \delta_t$$

where δ_t represents Gaussian noise with covariance matrix Q_t. In this case, Q_t is just a variance.

The Kalman filter can now be applied. The parameters we will use are as follows:

- Covariance matrix of motion equation: $R = \left[\begin{array}{cc} 0.2 & 0.05 \\ 0.05 & 0.1 \end{array} \right]$
- Variance of measurement equation: $Q = 0.5$

The standard form for generating measurements is as follows,

$$z_t = C_t x_t + \delta_t$$

In our case the matrix C_t is quite simple,

$$z_t = [1 \ 0] \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \delta_t$$

where δ_t represents Gaussian noise with covariance matrix Q_t. In this case, Q_t is just a variance.

The Kalman filter can now be applied. The parameters we will use are as follows:

- Covariance matrix of motion equation: $R = \begin{bmatrix} 0.2 & 0.05 \\ 0.05 & 0.1 \end{bmatrix}$
- Variance of measurement equation: $Q = 0.5$
- Mass of robot: $m = 1$

The standard form for generating measurements is as follows,

$$z_t = C_t x_t + \delta_t$$

In our case the matrix C_t is quite simple,

$$z_t = [1\ 0] \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \delta_t$$

where δ_t represents Gaussian noise with covariance matrix Q_t. In this case, Q_t is just a variance.

The Kalman filter can now be applied. The parameters we will use are as follows:

- Covariance matrix of motion equation: $R = \begin{bmatrix} 0.2 & 0.05 \\ 0.05 & 0.1 \end{bmatrix}$
- Variance of measurement equation: $Q = 0.5$
- Mass of robot: $m = 1$
- Time step: $\Delta t = 1$

DEMO IN MATLAB

Properties of Kalman filters:

- Kalman filters are highly efficient

Properties of Kalman filters:

- Kalman filters are highly efficient
    - Cost of matrix inversion on line 4: $O(k^{2.376})$

Properties of Kalman filters:

- Kalman filters are highly efficient
  - Cost of matrix inversion on line 4: $O(k^{2.376})$
  - Cost of multiplying $n \times n$ matrices: $O(n^2)$

Properties of Kalman filters:

- Kalman filters are highly efficient
  - Cost of matrix inversion on line 4: $O(k^{2.376})$
  - Cost of multiplying $n \times n$ matrices: $O(n^2)$
- Optimal for linear Gaussian systems...

Properties of Kalman filters:

- Kalman filters are highly efficient
  - Cost of matrix inversion on line 4: $O(k^{2.376})$
  - Cost of multiplying $n \times n$ matrices: $O(n^2)$
- Optimal for linear Gaussian systems...
- Unfortunately, most robotic systems are non-linear!

Properties of Kalman filters:

- Kalman filters are highly efficient
  - Cost of matrix inversion on line 4: $O(k^{2.376})$
  - Cost of multiplying $n \times n$ matrices: $O(n^2)$
- Optimal for linear Gaussian systems...
- Unfortunately, most robotic systems are non-linear!
- For non-linear systems you can linearize and apply the Extended Kalman Filter