

# Algorithms

Brian Zhu

July 27, 2020

Useful Notation:

Matrices - capital letters, ex:  $A$

Vectors - have arrow above, ex:  $\vec{x}$

scalars - lower case letters, ex:  $a$

a priori estimate - ex:  $x^-$

estimate of state after measurement update - ex:  $x_k$

## 1 Kalman Filter

### 1.1 Prediction (Time update)

1.  $x_t = Ax_{t-1} + Bu_{t-1}$  PREDICT STATE
  - $x_t$  - new state
  - $x_{t-1}$  - old state
  - $A$  - matrix which tells us how shape itself changes over that old state
  - $u_{t-1}$  - control value
  - $B$  - Matrix value of how control changes over that old state
2.  $P_t^- = AP_{t-1}A^T + Q$  PROJECT ERROR
  - $P_t^-$  - Prior before the corrector step ('-'superscript is a priori)
  - $AP_{t-1}A^T$  - what is this...
  - $Q$  - movement error (random Gaussian noise)

### 1.2 Corrector (Measurement update)

1. Compute  $K$  (Kalman Gain)  
 $K = P_t^- H^T (HP_t^- H^T + R)^{-1}$
2. Update Estimate  
 $\vec{x}_t = \vec{x}_t^- + K(\vec{z}_t - H\vec{x}_t^-)$ 
  - $(\vec{z}_t - H\vec{x}_t^-)$  - innovation

- This is the difference between the observation and the location of the Estimate. if we are where we say we are, then we should see the landmark at this location. But were not, we are seeing it at the observed measurement. This is the error and we are scaling it by the kalman filter and use it to correct the position that we have
- 3. Update Covariance  

$$P_t = (I - K_t H) P_t^-$$
  - make an observation of landmark, uncertainty shrinks

We can only use Kalman filter for problems where the update equation/system is linear, thus we move to extended kalman filter

## 2 Extended Kalman Filter (EKF)

### 2.1 Predict

1. Project State Forward  

$$x_t^- = g(x_{t-1}, u_{t-1})$$
2. Project Covariance  

$$P_t^- = G_t P_{t-1} G_t^T + Q$$

### 2.2 Correct

1. Calculate Kalman Gain (K)  

$$K_t = P_t^- H^T (H P_t^- H^T + Q)^{-1}$$
2. Correct  $\vec{x}_t^-$   

$$\vec{x}_t = \vec{x}_t^- + K \hat{z}$$
  - $\hat{z} = (\vec{z}_t - h(\vec{x}_t^-))$ 
    - $\vec{z}_t$  - Actual Measurement
    - $h(\vec{x}_t^-)$  - Measurement made by prediction
    - $\hat{z}$  - is error
  - multiply error by Kalman filter to adjust state
3. Correct  $P_t^-$   

$$P_t = (I - K_t H_t) P_t^-$$
  - $g$  and  $h$  are non linear equations that govern how the system works
    - $g$  is motion model of the robot, how the robot moves (probably non-linear)
    - $h$  is equation that tells us how to collect measurements. Produces  $\vec{z}_t$
  - $G$  and  $H$  are the derivatives of  $g$  and  $h$
  - Jacobian is derivative in all directions

## 3 2D Planar Robot

### 3.1 Pose, Control, Movement Variables

- $\vec{s}_t$  represents position(Pose) vector

$$\vec{s}_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

- $x$  and  $y$  measured in meters
- $\theta$  measured in radians

- $\vec{u}_t$  is control

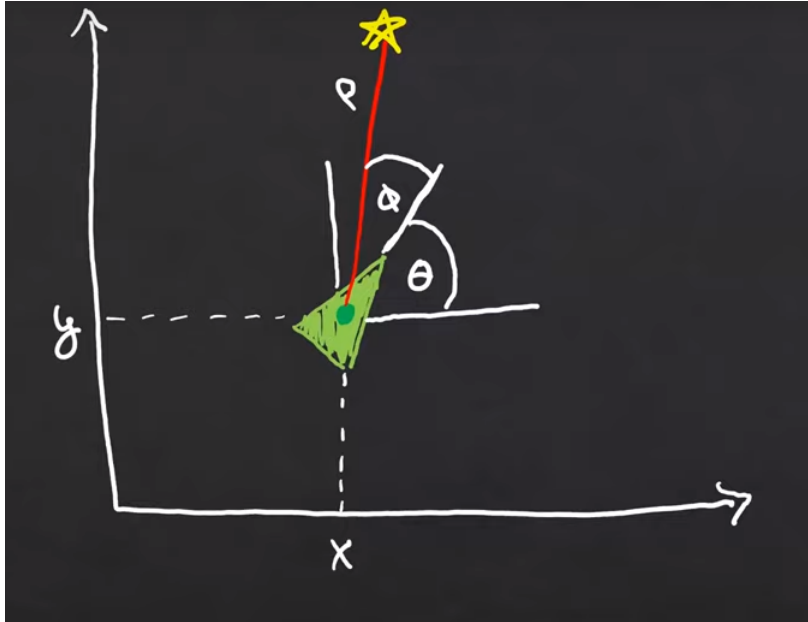
$$\vec{u}_t = \begin{bmatrix} v \\ \omega \end{bmatrix}$$

- $v$  measured in meters per second
- $\omega$  measured in radians per second

- $\vec{z}_t$  is measurement

$$\vec{z}_t = \begin{bmatrix} \rho \\ \phi \end{bmatrix}$$

- $\rho$ (range) measured in meters
- $\phi$ (bearing) measured in radians



## 3.2 Equations

Important to note that in robotics we will replace  $\vec{x}_t$  with  $\vec{s}_t$ . So going forward  $\vec{s}_t$  is position vector. Other equations prior to now were general format.

- $g(\vec{s}_t, \vec{u}_t) \rightarrow \vec{s}_{t+1}$ 
  - $\vec{s}_t$  - current position
  - $\vec{u}_t$  - current control
  - $\vec{s}_{t+1}$  - new position

### 3.2.1 Breaking down $\vec{s}_t$

- $x_t = x_{t-1} + v_x \Delta t$
- $y_t = y_{t-1} + v_y \Delta t$ 
  - IMPORTANT: Remember these now refer to individual components of  $\vec{s}_t$

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix} \Delta t$$

$$\vec{s}_t = \vec{s}_{t-1} + \vec{u}_t \Delta t$$

- $\vec{s}_{t-1} + \vec{u}_t \Delta t = g(\vec{s}_t, \vec{u}_t)$

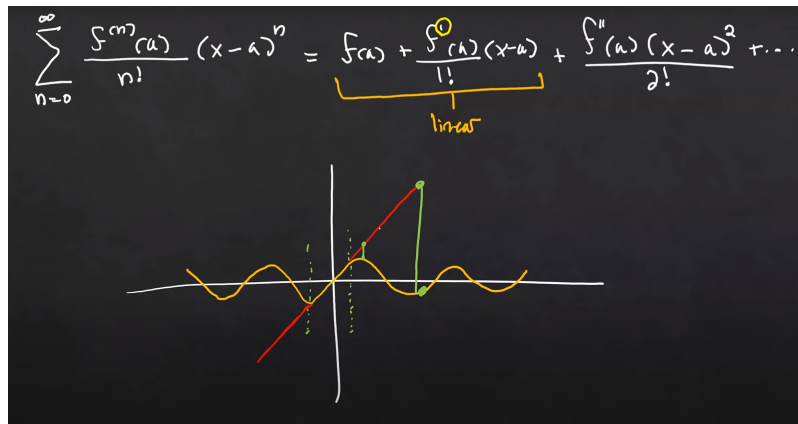
## 3.3 Jacobian

G is a Jacobian of  $g(\vec{s}_t, \vec{u}_t)$

- $G_t = \nabla_s g(\vec{s}_t, \vec{u}_t)$
- $H_t = \nabla_s h(\vec{s}_t)$

### 3.3.1 Example gradient

$$\begin{bmatrix} x & +y^2 \\ x^2 & -y \end{bmatrix} \rightarrow \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 1 & 2y \\ 2x & 1 \end{bmatrix}$$



- notice how the Taylor series can approximately estimate a portion of a non-linear function using a linear function
- It uses the first derivative, and as long as we stay within the bounds as shown above, we can accurately model with a linear function
- Now that its linear, that means we can use Kalman filter and be optimal yet again

## 4 Particle Filters

- particles are hypotheses about the system  $(x, y, \theta)$ 
  - remember the wheel chair video with the arrows on the ground, each arrow is a particle(hypothesis)
- Take average  $x, y, \theta$  and you get where the robot is. Analogous to  $\vec{s}_t$  from kalman filter  
Distribution of arrows is analogous to  $\vec{P}_t$ , uncertainty to where robot is
- When particles are spread apart, low confidence to where it is, opposite means high confidence to where it is
- When robot moves, we move the particles as well.
  - Apply motion model to particles to propagate robots motions
  - Particles will spread out
- When robot picks up a landmark, it will get a range and a bearing measurement  $\rho, \phi$ 
  - Then each particle will test against those measurements
  - Given these tests, we will eliminate those whose measurements tell us that there is a low probability of the particle being right

- Then we will duplicate the remaining particles to make sure that we always maintain a certain number of particles (kind of reminds me of genetic algorithm)

## 4.1 Algorithm Outline

1. Sample - Sample M particles from proposal Distribution
2. Weight - assign score (weight) to each particle based on how well it matches posterior dist.
3. Resample - choose particles for the new set proportional to their weights

## 4.2 Algorithm

- Input:  $Y_{t-1}, u_t, z_t$ 
  - $Y_{t-1} = [s_{1,t-1}, s_{2,t-1}, \dots, s_{n,t-1}]^T$   
Set of all the particles. If there are n different particles, then  $Y_{t-1}$  has all the particles
  - $u_t = [v_t, \omega_t]^T$
  - $z_t = [\rho_t, \phi_t]^T$
- Output:  $Y_t$ , M particles (new set of particles)

### 4.2.1 Pseudo-Code

let  $y[m]$  be  $y_{t-1}^{[m]}$  where m is the mth particle  
 u be  $u_t$   
 z be  $z_t$   
 w[m] be  $w_t^{[m]}$   
 Y be  $Y_t$

```

for m in M do:
    get y[m]
    Project it forward with u + noise
    Weight prediction with z + noise
end

for m in M do:
    draw with probability of w[m]
    add to new particle set Y
end

```

- We want noise so particles vary, if particles are the same then we don't have any probability about it being anywhere else

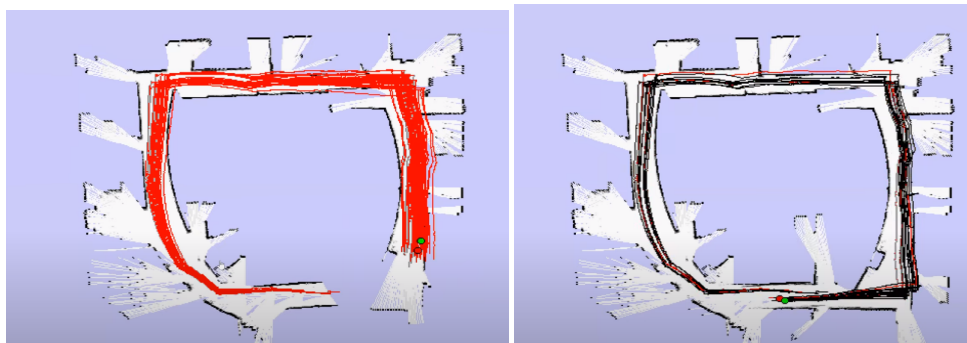
### 4.3 Tips and Strategies for Resampling

1. Dont resample too frequently
  - Pause when no controls or observations
2. Dont resample independently
  - Stratified Random Sampling
3. Add random particles after resampling
  - Helps with robustness
  - If robot has diverged, not enough randomness can just make it diverge into oblivion
  - A random particle can be lucky if the robot has diverged and help get it back on track
  - Solves "kidnapped robot problem"

## 5 FastSLAM

### 5.1 How it works

- Use particles (robots position/movement plus some noise), along with the robots own readings, to create a map
- Each particle will also be building its own map of the state
- When particles reach a point that they have seen before, all improbable/impossible particles will be eliminated
- Much more consistent map, causes better results



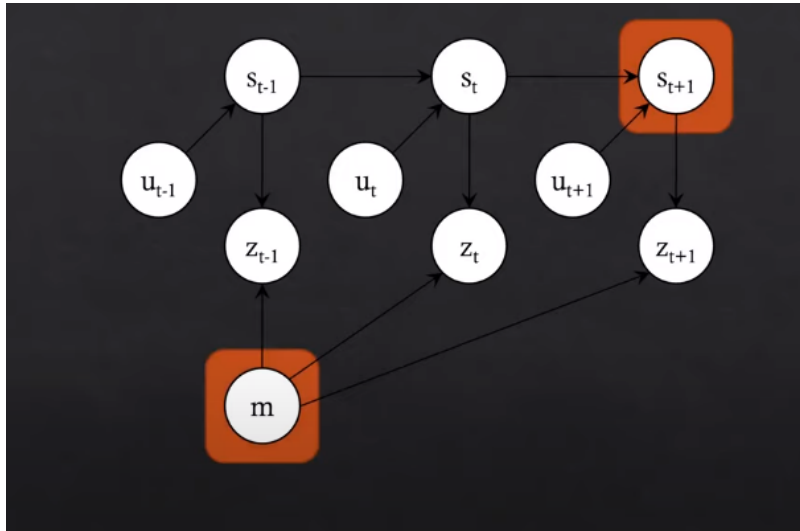
Particles that cannot have existed are eliminated

## 5.2 What we are trying to estimate

### 5.2.1 Online SLAM

$$p(s_t, m | z_{1:t}, u_{1:t})$$

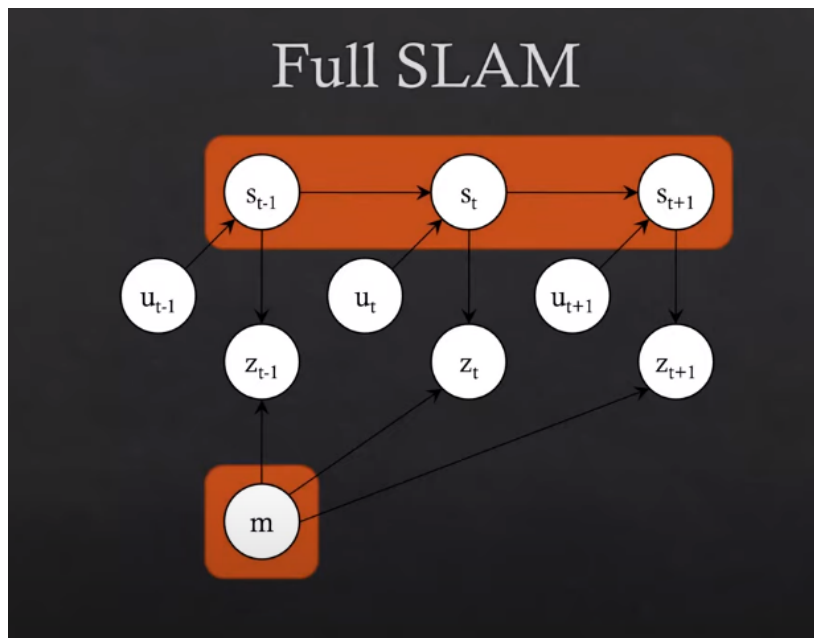
- Probability of the pose and the map, given the history of observations and history of transformations of the pose
  - pose:  $s_t$
  - map:  $m$
  - history of observations:  $z_{1:t}$
  - history of transformations:  $u_{1:t}$



- We start with the previous pose ( $s_{t-1}$ ) and the map,  $m$
- Then we have an observation ( $z_{t-1}$ ), which has to be consistent with the previous pose and the map
- Then we apply a control ( $u_{t-1}$ ) to  $s_{t-1}$  to get  $s_t$
- Then  $s_t$  will make an observation,  $z_t$ , from the world which is drawn from the map
- Then we apply a control ( $u_t$ ) to  $s_t$  to get  $s_{t+1}$
- Then  $s_{t+1}$  will make an observation,  $z_{t+1}$ , from the world which is drawn from the map
- Repeats
- What we want in the end, as the outputs, is the map and the final pose



### 5.2.2 Full SLAM



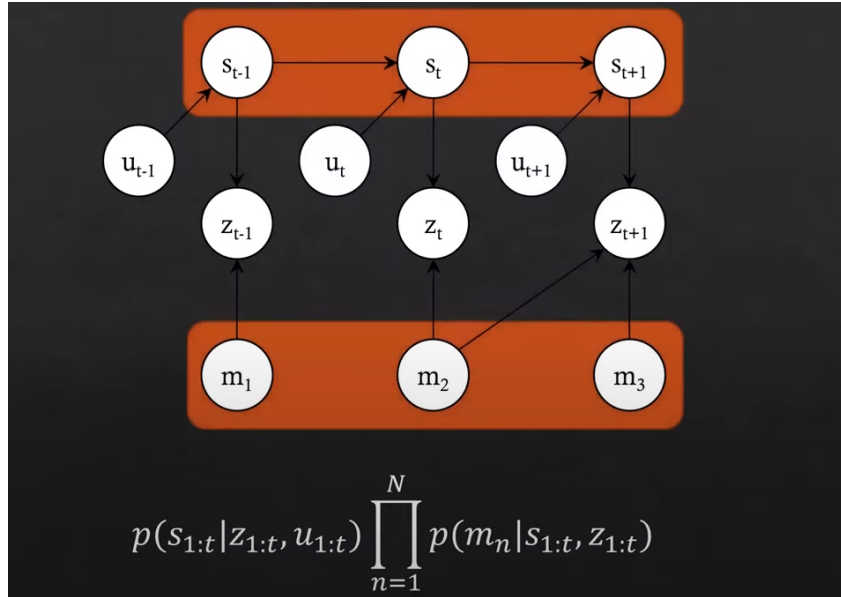
- Not only estimates the final pose of the robot, it estimates the entire history
  - History represented by the black lines on the picture of robot using SLAM

### 5.2.3 FastSLAM

- $p(m_n | s_{1:t}, z_{1:t})$ 
  - probability of a map given the previous states and observations of the robot
    - \*  $m_n$  is a specific particles map, given that there are  $N$  particles
    - \*  $s_{1:t}$  is previous states
    - \*  $z_{1:t}$  is previous observations
- $p(s_{1:t} | z_{1:t}, u_{1:t})$ 
  - probability of the state given the position and controls of the robot

$$\prod_{n=1}^N$$

Just Sigma summation but multiply not add



- We start with the previous pose ( $s_{t-1}$ ) and the map,  $m$
- Then we have an observation ( $z_{t-1}$ ), which has to be consistent with the previous pose and the map
- Then we apply a control ( $u_{t-1}$ ) to  $s_{t-1}$  to get  $s_t$
- Then  $s_t$  will make an observation,  $z_t$ , from the world and then generates a map2, updates the map
- Then we apply a control ( $u_t$ ) to  $s_t$  to get  $s_{t+1}$
- Then  $s_{t+1}$  will make an observation,  $z_{t+1}$ , from the world and then generates a map3
  - At time  $z_{t+1}$ , map element 2 and 3 were both observed
- Repeats
- **Actually estimating the entire state and all of the map elements**

### 5.3 Terminology

- Robot Pose

$$s_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

- Observation

$$z_t = \begin{bmatrix} \rho \\ \phi \end{bmatrix}$$

- Control

$$u_t = \begin{bmatrix} v \\ \omega \end{bmatrix}$$


- Map Feature:  $\mu$  is like an x and y(position of landmark).  $\Sigma$  describes uncertainty in the pose. Covariance matrix

$$m_t = \begin{bmatrix} \mu \\ \Sigma \end{bmatrix}$$

## 5.4 Algorithm: Fast Slam

- On Input:  $(Y_t, u_t, z_t)$ 
  - $Y_t$  is a set of particles
  - $u_t$  is control
  - $z_t$  is the observation
- Output:  $Y_{t+1}$ 
  - New set of particles
- Do M times
  - Predict - Sample particle from motion model
  - Observe - Update N landmark EKF's
    - \* using  $z_t$  observe N landmarks
    - \* Update EKF's in the same way we do a typical EKF update. Go through and update each of the EKF's, moving  $\mu$  and updating the covariance matrix(typically making it smaller)
  - Weight - Importance weight new particle. Particles of low weight are not resampled
- Resample - M particles (with replacement) proportional to  $w^{[k]}$ 
  - "with replacement" means can take the same particle multiple times
  - Before and After we have M particles. Some particles at  $t + 1$  will be duplicate particles but those are going to differentiate themselves in the next step where we apply a different control to each particle. (Have control, then apply noise)
  - Sample with a weight proportional to the weight of the  $k^{th}$  particle

#### 5.4.1 Every Particle is a Hypothesis



$$Y_t^{[k]} = \left\langle s_{1:t}^{[k]}, \left\langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]} \right\rangle, \dots, \left\langle \mu_{N,t}^{[k]}, \Sigma_{N,t}^{[k]} \right\rangle, w^{[k]} \right\rangle$$

- $Y_t^{[k]}$  -  $k^{th}$  particle at time  $t$
- $s_{1:t}^{[k]}$  - path of particle  $k$ , the  $k^{th}$  particle's history of positions. Like an array of positions,  $x, y, \theta$  for every time step
- $\langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]} \rangle$  - N Map features. Array of map features
  - $\mu_{1,t}^{[k]}$  - Expected pose of landmark1 at time  $t$  according to particle  $k$
  - $\Sigma_{1,t}^{[k]}$  - uncertainty in pose of landmark1 at time  $t$  according to particle  $k$
- $w^{[k]}$  - weight of the  $k^{th}$  particle; Confidence in particle  $k$

### 5.4.2 Prediction

**FastSLAM 1.0 - Predict**  
Sample from motion model

$$s_t^{[k]} \sim p(s_t | s_{t-1}^{[k]}, u_t)$$

Add control noise

$$\begin{aligned}\hat{v} &= v + \text{randn}(\alpha_1 v + \alpha_2 \omega) \\ \hat{\omega} &= \omega + \text{randn}(\alpha_3 v + \alpha_4 \omega)\end{aligned}$$

Propagate motion

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} -\frac{\hat{v}}{\hat{\omega}} \sin \theta + \frac{\hat{v}}{\hat{\omega}} \sin(\theta + \omega \Delta t) \\ \frac{\hat{v}}{\hat{\omega}} \cos \theta + \frac{\hat{v}}{\hat{\omega}} \cos(\theta + \omega \Delta t) \\ \hat{\omega} \Delta t \end{bmatrix}$$

### 5.4.3 Observation

**FastSLAM 1.0 – Observe New Feature**  
Observation Model

$$h(\mu_j, \hat{s}_t) = \begin{bmatrix} \sqrt{(\mu_{j,x} - \hat{s}_{t,x})^2 + (\mu_{j,y} - \hat{s}_{t,y})^2} \\ \text{atan2}(\mu_{j,y} - \hat{s}_{t,y}, \mu_{j,x} - \hat{s}_{t,x}) \end{bmatrix}$$

Feature Location

$$\mu_{j,t}^{[k]} = h^{-1}(z_t, s_t^{[k]})$$

Jacobian with respect to feature

$$H = \nabla_{m_j} h(s_t^{[k]}, \mu_{j,t}^{[k]})$$

Feature Covariance

$$\Sigma_{j,t}^{[k]} = H^{-1} Q_t (H^{-1})^T$$

#### 5.4.4 Observe Old

### FastSLAM 1.0 – Observe Old Feature

$$\hat{z} = h(\mu_{j,t-1}^{[k]}, s_t^{[k]})$$

Predict measurement

$$H = \nabla_{m_j} h(s_t^{[k]}, \mu_{j,t-1}^{[k]})$$

Jacobian w.r.t. feature

$$Q = H \Sigma_{j,t-1}^{[k]} H^T + Q_t$$

Measurement Covariance

$$K = \Sigma_{j,t-1}^{[k]} H^T Q^{-1}$$

Kalman Gain

$$\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]} + K(z_t - \hat{z})$$

New feature mean

$$\Sigma_{j,t}^{[k]} = (I - KH) \Sigma_{j,t-1}^{[k]}$$

New feature covariance