

MORAI

Localization & Mapping
- Filter based method

2021-04-20

목차

1. Kalman Filter
2. Extended KF 이론
3. Extended KF 실습
4. Particle Filter 이론
5. Particle Filter 실습
6. Landmark based localization with PF

Estimation

Kalman Filter 이론

Estimation

➤ Kalman Filter

- 1960년대 루돌프 칼만이 발표한 재귀 필터 알고리즘
- 한 스텝 이전의 추정값과, 현재의 측정값, 모델 파라미터만 있으면 어떤 디바이스에 탑재 가능.
- 아폴로 프로젝트에서 trajectory estimation 등에 적용.

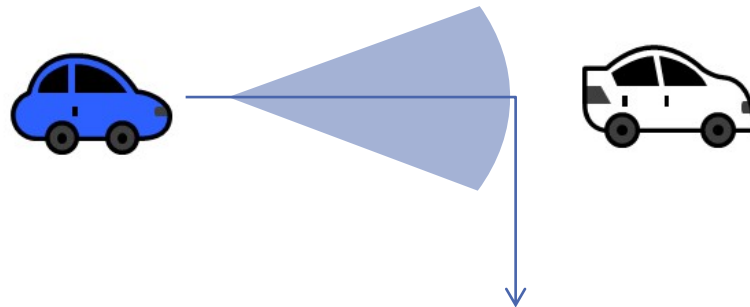
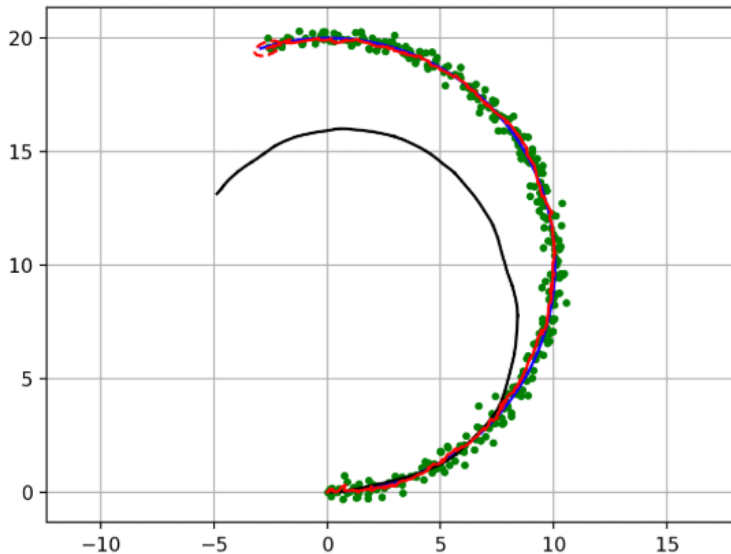
Q: It was often stated that the TRS-80 Model I was more powerful than the computer used by the Apollo lunar module. As someone who actually worked on the Apollo program, do you think that was true?

A: Yes, absolutely. I've heard it said that a modern digital watch is more powerful than the Apollo flight computer. When you think about it, the remarkable thing is that we had a flight computer at all. Consider: There was no such thing as RAM chips, much less CPUs. The Apollo computer used 2k of magnetic core RAM and 36k wire rope (ROM) memory. The CPU was built from ICs, but not the kind we think of today. They were, in fact, the same Fairchild RTL chips I fell in love with. Clock speed was under 100 kHz. Compare that to the 16k ROM, 16-48k RAM, and 2 MHz clock of the TRS-80.

Estimation

➤ Kalman Filter 사용분야

- GPS/IMU 보정.
- ADAS : 앞차간 간격 추정, 차선 추적, 속도 추정 등등.
- VR, AR.



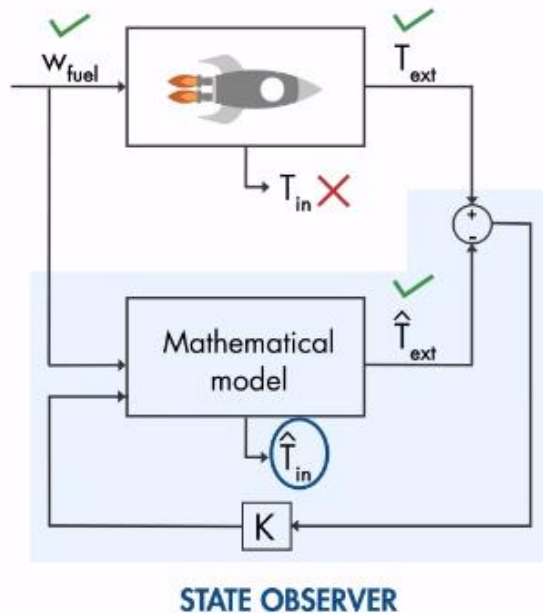
Radar + IMU -> 거리추정

<https://pythonrobotics.readthedocs.io/en/latest/modules/localization.html>

Estimation

➤ Kalman Filter 필요성

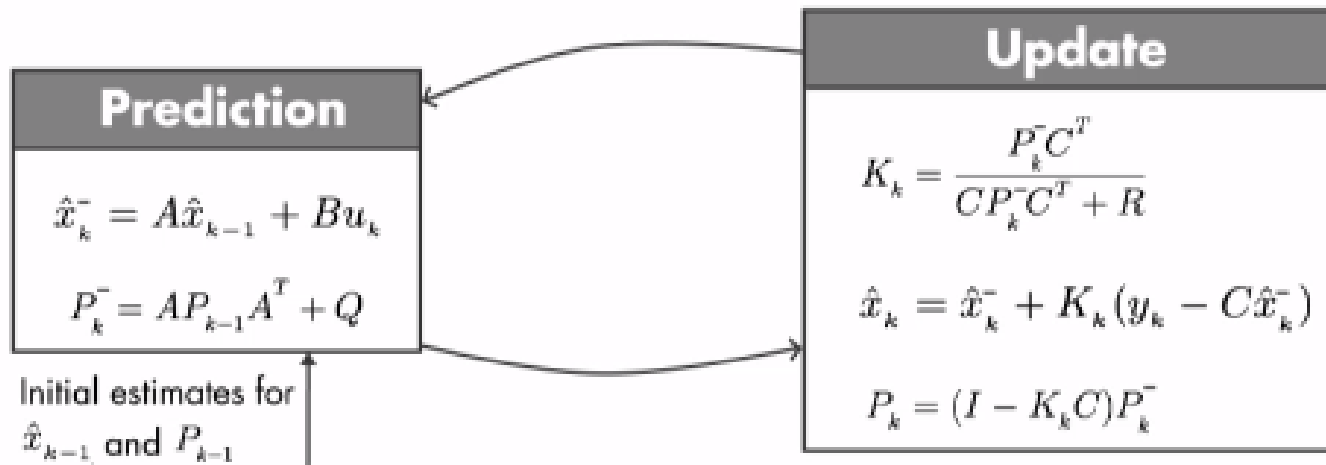
- 센서는 항상 노이즈가 낀다.
- Low pass filter는 phase lag을 유발.
- 수학적 모델링이 현실이랑 종종 잘 안 맞음.



<https://kr.mathworks.com/videos/understanding-kalman-filters-part-2-state-observers-1487694734527.html>

➤ Kalman Filter 단계

- 칼만필터는 보통 두가지 단계로 나뉜다.
- 예측 : 이전의 추정값과 현재의 입력값, 가정된 노이즈 분산값으로 현재 추정값 예측.
- 보정 : 예측된 값을 센서에서 나오는 측정 값으로 보정



\hat{x} : 추정되는 상태의 평균 (속도, 거리 등등. 정답은 모름)

P : 추정되는 상태의 공분산 (속도, 거리 등등. 정답은 모름)

z : 측정 (센서로부터 나오는 값. 노이즈 존재)

Estimation

예측(prediction)

- 이전 단계의 추정값(\hat{x}_{k-1})과 입력(u_k)을 가지고 상태 예측(\hat{x}_k^-, P_k^-).
- 예측 모델은 개발자가 정의해줘야 한다.
- 가정 1: 예측 모델과 측정 모델이 linear할 경우
- 가정 2: 예측 모델과 측정 모델이 정규분포를 따를 경우

$$x_k = [pos, vel] = [p_k, v_k]^T$$

$$u_k = [accel]$$

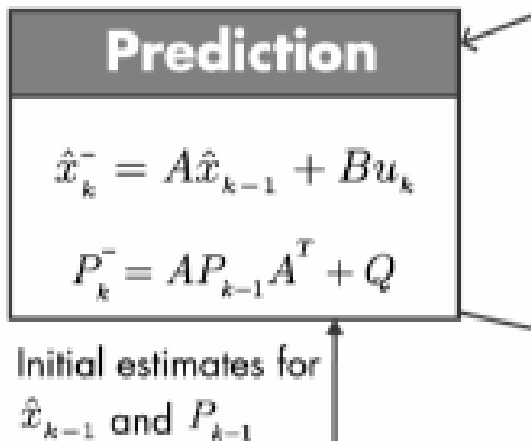


$$x_{k+1} = f(x_k, u_k) + w_k$$



예측 모델의 불확실성 노이즈 :

$$w_k \sim \mathcal{N}(0, Q)$$

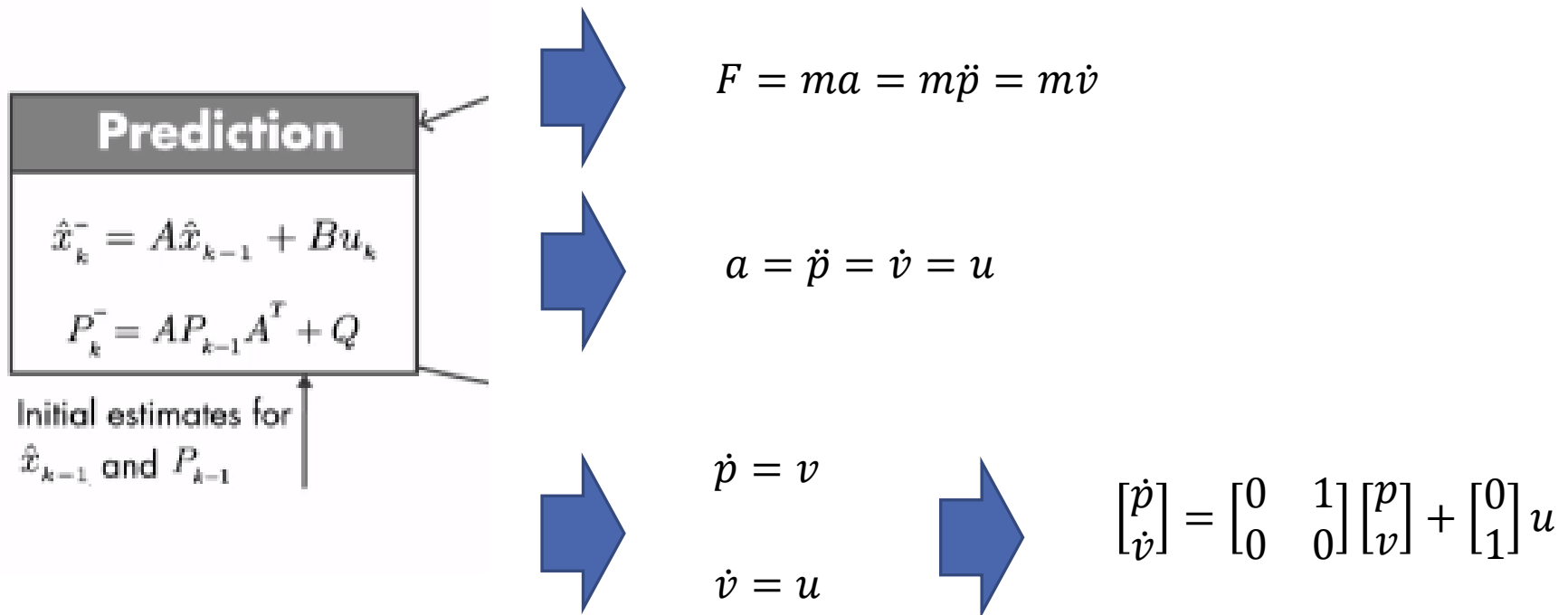


Estimation

예측(prediction)

- 이전 단계의 추정값(\hat{x}_{k-1})과 입력 (u_k) 을 가지고 상태 예측(\hat{x}_k^-, P_k^-).
- 예측 모델은 개발자가 정의해줘야 한다.

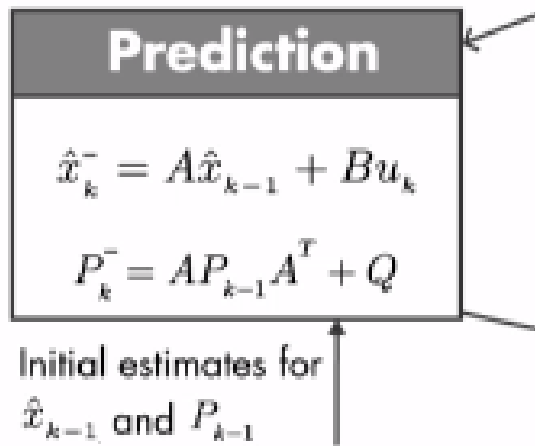
$$x_k = [pos, vel] = [p_k, v_k]^T$$



Estimation

예측(prediction)

- 이전 단계의 추정값($\hat{x}_{(k-1)}$)과 입력 (u_k) 을 가지고 상태 예측(\hat{x}_k^-, P_k^-).
- 예측 모델은 개발자가 정의해줘야 한다.



$$x_k = [pos, vel] = [p_k, v_k]^T$$

$$\dot{x} = (x_{k+1} - x_k) / \Delta t$$

$$x_k = x_{k-1} + \dot{x}\Delta t$$


$$\begin{bmatrix} p \\ v \end{bmatrix}_{k+1} = \begin{bmatrix} p \\ v \end{bmatrix}_k + \Delta t \left(\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ v \end{bmatrix}_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \right)$$

$$\begin{aligned} \begin{bmatrix} p \\ v \end{bmatrix}_{k+1} &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ v \end{bmatrix}_k + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_k \\ &= A \begin{bmatrix} p \\ v \end{bmatrix}_k + Bu_k \end{aligned}$$

➤ 예측(prediction)

- 상태의 공분산 계산.
- 정규분포를 따르는 변수 y 가 있으면 $z = ay + b$ 의 평균과 분산은 아래와 같다.


$$y \sim \mathcal{N}(\mu, \sigma^2)$$


$$z = ay + b \sim \mathcal{N}(a\mu + b, a^2\sigma^2)$$

- 벡터와 행렬로 확장시켜보자.

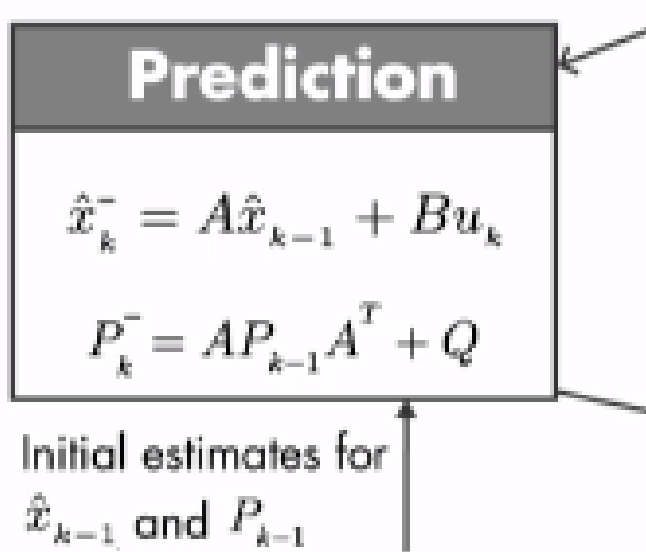
스칼라에서 벡터와 매트릭스로 확장

$$\vec{y} \sim \mathcal{N}(\vec{\mu}, \Sigma)$$


$$\vec{z} = A\vec{y} + \vec{b} \sim \mathcal{N}(A\vec{\mu} + \vec{b}, A\Sigma A^T)$$

Estimation

- 예측(prediction)
 - 모델 확정.



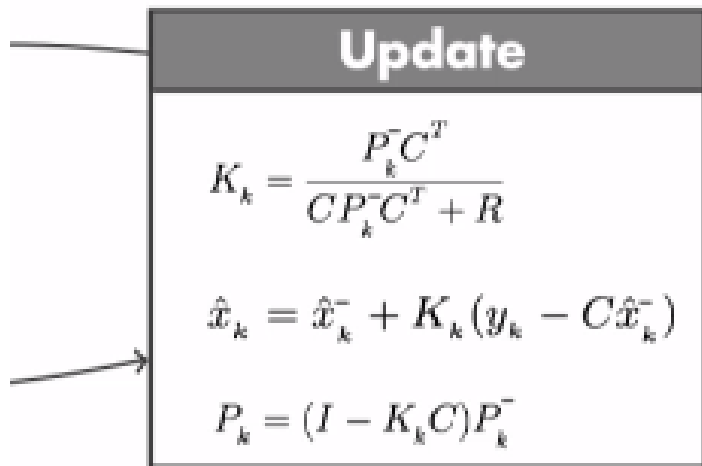
$$\begin{bmatrix} p \\ v \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ v \end{bmatrix}_k + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_k$$

$$P_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} P \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}^T + Q_k$$

Estimation

➤ 보정(correction)

- 상태 예측(\hat{x}_{k^-}, P_{k^-})한 결과에 측정값(z_k)을 반영해서 보정하고 상태 추정 (\hat{x}_k, P_k) 확정.
- 측정 모델은 현재 쓸수 있는 센서를 기반으로 정의해줘야 한다.
- 가정 1: 예측 모델과 측정 모델이 linear할 경우
- 가정 2: 예측 모델과 측정 모델이 정규분포를 따를 경우



IMU -> 속도

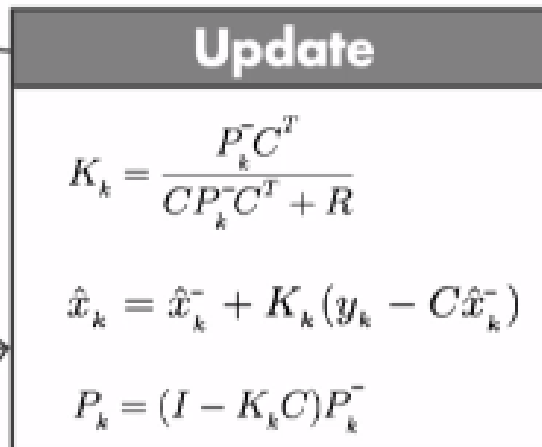
$$z_k = g(x_k) + n_k$$

측정모델의 불확실성 노이즈 : $v_k \sim \mathcal{N}(0, R)$

Estimation

➤ 보정(correction)

- 상태 예측(\hat{x}_{k^-}, P_{k^-})한 결과에 측정값(z_k)을 반영해서 보정하고 상태 추정 (\hat{x}_k, P_k) 확정.
- 측정 모델은 현재 쓸수 있는 센서를 기반으로 정의해줘야 한다.
- GPS로 위치만 알 수 있다 가정하자.



GPS -> 속도

$$\begin{aligned} z_k &= g(x_k) + n_k = p_k + n_k \\ &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} p \\ v \end{bmatrix}_k + n_k \end{aligned}$$

Estimation

➤ 보정(correction)

- 상태 예측(\hat{x}_{k^-} , P_{k^-})한 결과에 측정값(z_k)을 반영해서 보정하고 상태 추정 (\hat{x}_k , P_k) 확정.
- Kalman gain 계산 : 상태 예측(\hat{x}_{k^-} , P_{k^-})에 측정값(z_k)을 반영해서 얼마나 반영해줄지 결정.

Update

$$K_k = \frac{P_k^- C^T}{C P_k^- C^T + R}$$

$$\hat{x}_k = \hat{x}_k^- + K_k (y_k - C \hat{x}_k^-)$$

$$P_k = (I - K_k C) P_k^-$$

$$\hat{x}_k^- = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \hat{x}_{k-1} + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_k$$

$$P_k^- = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} P_{k-1} \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}^T + Q_k$$

$$= P_k^- [1 \quad 0]^T ([1 \quad 0] P_k^- [1 \quad 0]^T + R_k)^{-1} K_k$$

Estimation

➤ 보정(correction)

- Kalman gain을 사용하여, 최종 상태 추정 (\hat{x}_k , P_k) 확정.

Update

$$K_k = \frac{P_k^- C^T}{C P_k^- C^T + R}$$

$$\hat{x}_k = \hat{x}_k^- + K_k (y_k - C \hat{x}_k^-)$$

$$P_k = (I - K_k C) P_k^-$$

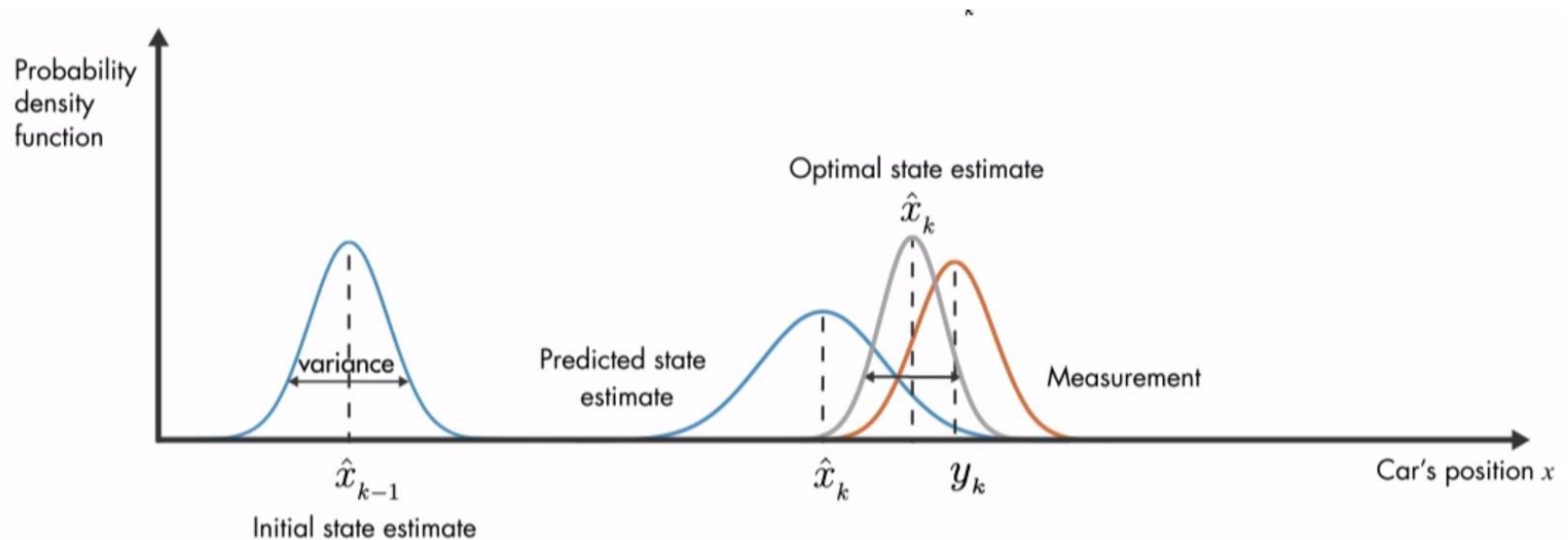
$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - [1 \ 0] \hat{x}_k^-)$$

$$P_k = (I - [1 \ 0] K_k) P_k^-$$

Estimation

➤ 1d example

- 1차원상에서 움직이는 모델에 대한 예측, 보정 표현.



<https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/#mathybits>

Estimation-KF

1d Kalman Filter example implementation

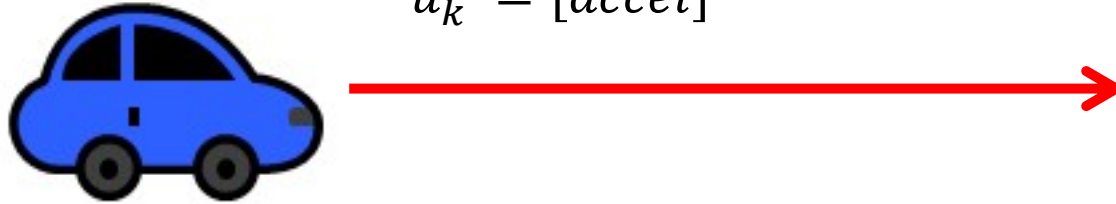
Estimation

➤ 1d example

- 차가 1차원 방향으로만 움직인다.
- 맞바람으로 인한 저항 등이 노이즈로서 작용한다고 가정.
- 위치만 측정 가능하며, 측정 노이즈가 존재

$$x_k = [pos, vel] = [p_k, v_k]^T$$

$$u_k = [accel]$$



$$x_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ v \end{bmatrix}_k + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_k + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} w_k$$

$$z_k = [1 \quad 0] \begin{bmatrix} p \\ v \end{bmatrix}_k + n_k$$

➤ 1d example

- Python code

```
class one_dim_env(object):
    def __init__(self, w_sig=0.5, v_sig=1, T=0.1):
        self.T = T
        self.A = np.array([[0,1],[0,0]])
        self.B = np.array([[0],[1]])

        self.Bw = np.array([[0],[1]])

        self.Ad = np.identity(2) + self.T * self.A
        self.Bd = self.T * self.B
        self.Bwd = self.T * self.Bw

        self.C = np.array([[1, 0]])

        self.x=np.array([[0],[10]])

        self.w_sig =w_sig
        self.v_sig =v_sig

    def internal_step(self, u):
        self.x=np.matmul(self.Ad, self.x) + self.Bd*u + self.Bwd*self.w_sig*np.random.randn(1)

    def external_step(self, u):
        self.internal_step(u)
        z = np.matmul(self.C, self.x) + self.v_sig*np.random.randn(1)
        return z
```

➤ 1d example

- KF code

```
class KALMANFilter(object):

    def __init__(self, T=0.1, Q=np.diag([0, 0.25]), R=1):

        self.T = T
        self.A = np.array([[0,1],[0,0]])
        self.B = np.array([[0],[1]])

        self.Ad = np.identity(2) + self.T * self.A
        self.Bd = self.T * self.B

        self.C = np.array([[1, 0]])

        self.x_hat = np.array([[0],[0]])
        self.P = np.diag([10, 10])

        self.Q = Q
        self.R = R/self.T
```

Estimation

➤ 1d example

- KF code

```
def prediction(self, u):  
  
    self.x_hat = np.matmul(self.Ad, self.x_hat) + self.Bd*u  
    self.P = np.matmul(np.matmul(self.Ad, self.P), self.Ad.T) + self.Q  
  
def measurement(self, z):  
  
    S = np.matmul(self.C, np.matmul(self.P, self.C.T)) + self.R  
    K = np.matmul(np.matmul(self.P, self.C.T), np.linalg.inv(S))  
    self.x_hat = self.x_hat + np.matmul(K, (z - self.C*self.x_hat))  
    self.P = np.matmul(np.identity(2) - np.matmul(K,self.C), self.P)
```

$$\hat{x}_k^- = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \hat{x}_{k-1} + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_k$$
$$P_k^- = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} P_{k-1} \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}^T + Q_k$$

$$K_k = \frac{P_k^- [1 \quad 0]^T}{[1 \quad 0] P_k^- [1 \quad 0]^T + R_k}$$
$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - [1 \quad 0] \hat{x}_k^-)$$
$$P_k = (I - [1 \quad 0] K_k) P_k^-$$

Estimation

➤ 1d example

- KF code

```
x_hat_hist = []
P_hist = []
z_hist = []

env = ONE_DIM_Env()
estimator = KALMANFilter()

u = np.concatenate([np.ones((30,)), np.zeros((40,)), -0.5*np.ones((30,))])

for i in range(100):

    #env steps
    z = env.external_step(u[i])

    #filter prediction
    estimator.prediction(u[i])

    #filter correction
    estimator.measurement(z)

    #record the process
    x_hat_hist.append(estimator.x_hat)
    P_hist.append(estimator.P)
    z_hist.append(z)
```

Estimation

➤ 1d example

- KF code

```
pos_hat = np.matmul(estimator.C, np.hstack(x_hat_hist)).squeeze(0)
vel_hat = np.hstack(x_hat_hist)[1, :]
sig_p = np.sqrt(np.vstack(P_hist).reshape([-1, 2, 2]))[:,0,0])
sig_v = np.sqrt(np.vstack(P_hist).reshape([-1, 2, 2]))[:,1,1])
z_np = np.hstack(z_hist).squeeze()
v_np = np.hstack(v_actual).squeeze()
con = 3

plt.figure(figsize=(20, 20))

plt.subplot(211)
plt.plot(np.arange(0, 10, 0.1), pos_hat, 'k-')
plt.plot(np.arange(0, 10, 0.1), pos_hat-con*sig_p, 'r--')
plt.plot(np.arange(0, 10, 0.1), pos_hat+con*sig_p, 'r--')
plt.plot(np.arange(0, 10, 0.1), z_np, 'b-')
plt.ylabel('position')
plt.xlabel('time')
```


Estimation

➤ 1d example

- KF code

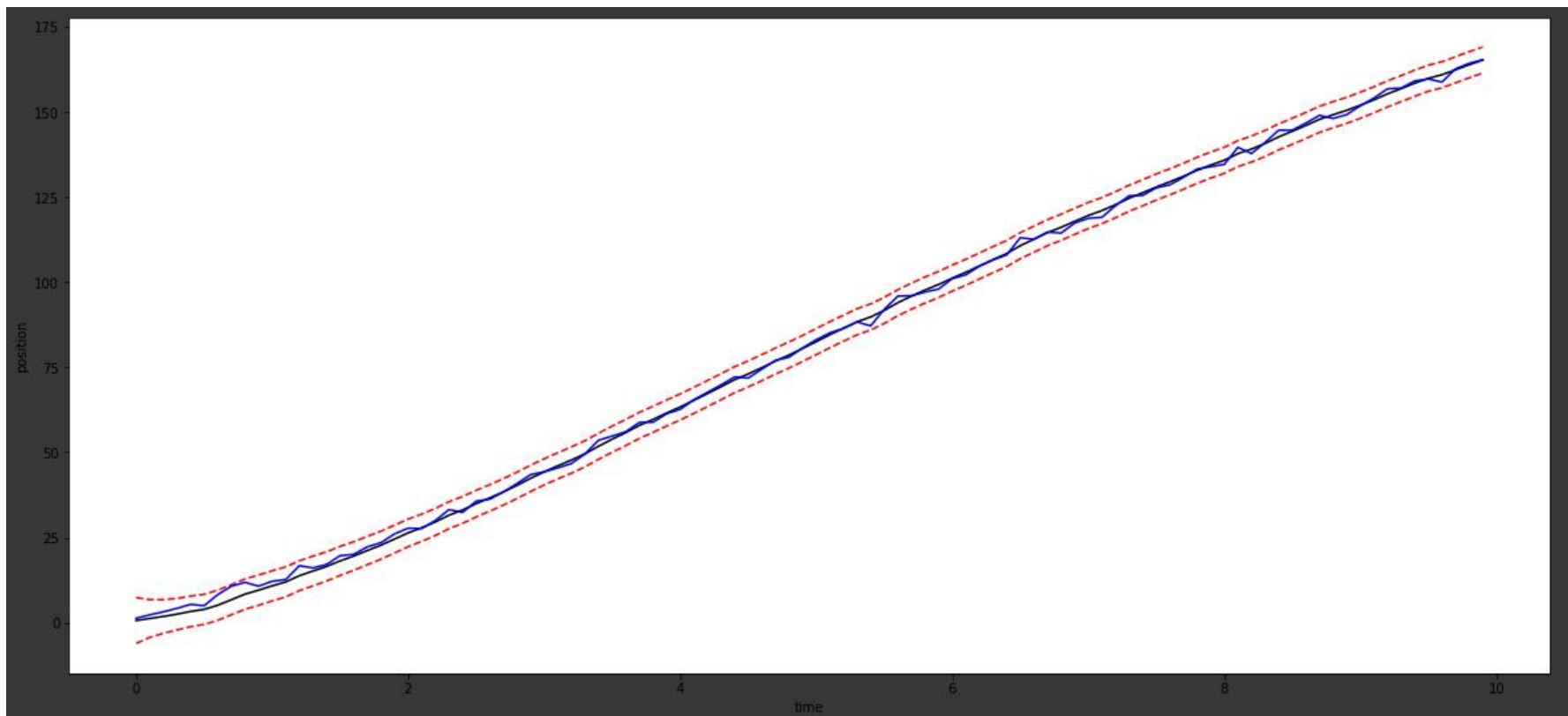
```
plt.subplot(212)
plt.plot(np.arange(0, 10, 0.1), vel_hat, 'k-')
plt.plot(np.arange(0, 10, 0.1), vel_hat-con*sig_v, 'r--')
plt.plot(np.arange(0, 10, 0.1), vel_hat+con*sig_v, 'r--')
plt.plot(np.arange(0, 10, 0.1), v_np, 'b-')
plt.plot(np.arange(0, 9.9, 0.1), (z_np[1:]-z_np[:-1])/0.1, 'g-')
plt.ylabel('velocity')
plt.xlabel('time')

plt.show()
```

Estimation

➤ 1d example

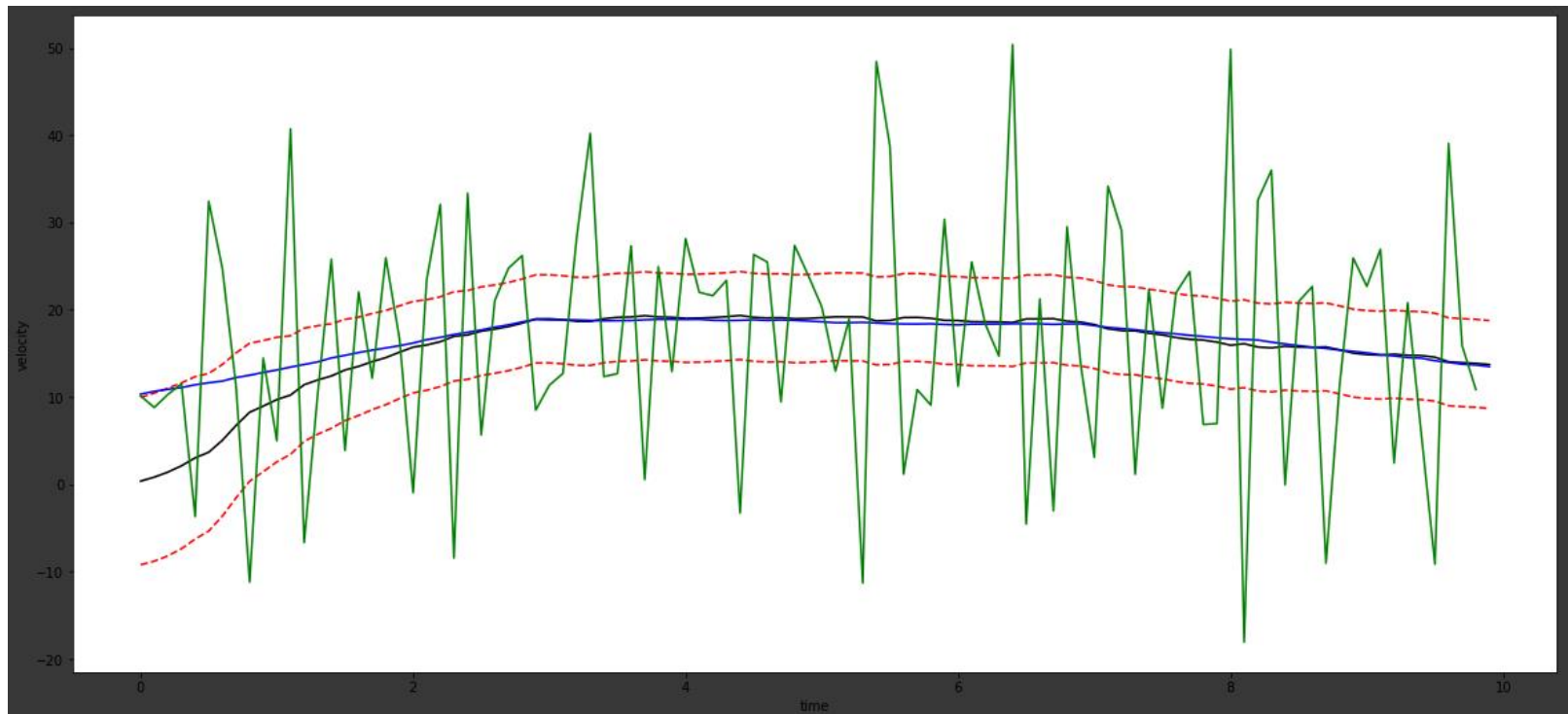
- 1차원상에서 움직이는 모델에 대한 예측, 보정 표현.



Estimation

➤ plotting

- 파란색 : 실제 환경에서의 속도.
- 검은색 : 칼만 필터로 추정한 속도.
- 초록색 : 단순히 이전 스텝과의 위치 차이로 구한 속도.



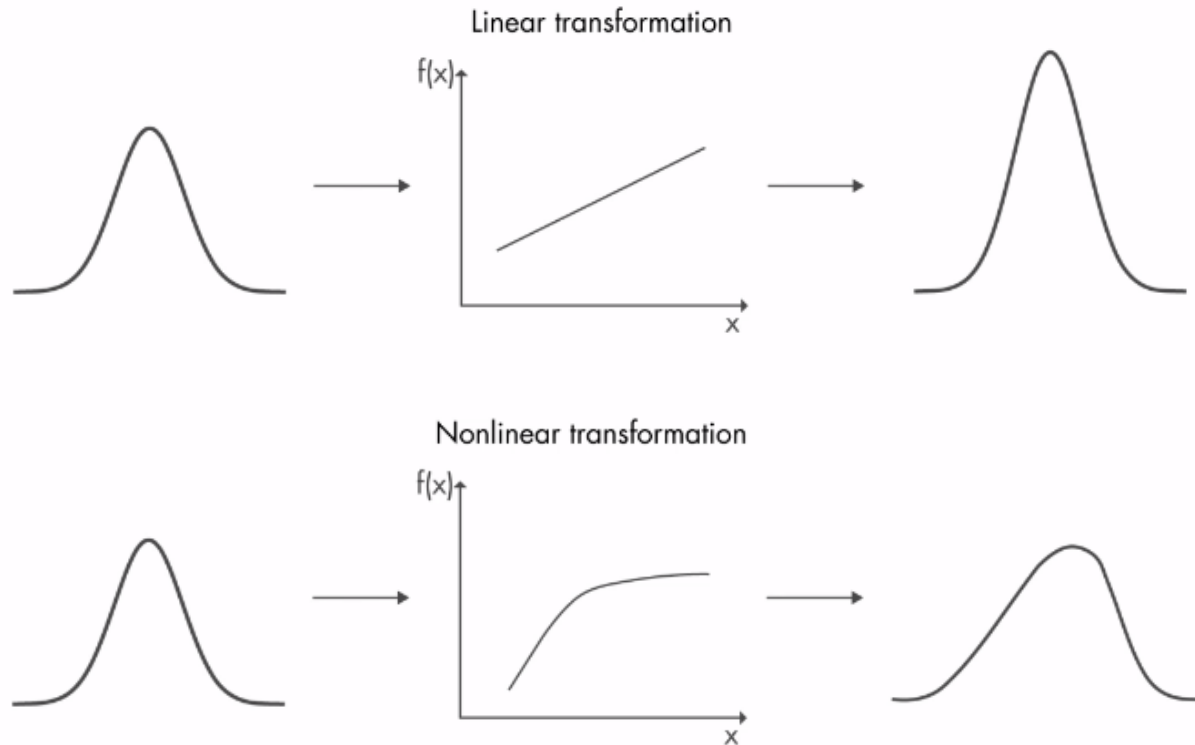
Estimation-EKF

Extended Kalman Filter

Estimation

➤ Kalman Filter의 문제점

- 비선형 모델에선 결과 분포가 Gaussian으로 유지되지 않는다.

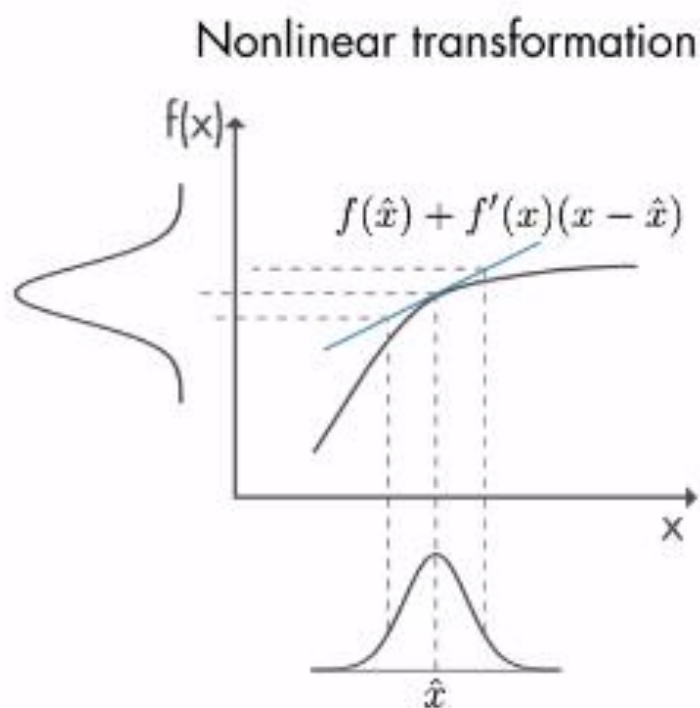


<https://kr.mathworks.com/videos/understanding-kalman-filters-part-5-nonlinear-state-estimators-1495052905460.html>

Estimation

➤ Extended Kalman Filter

- 자코비안 행렬을 이용한 선형화.
- 실제 분포와는 차이가 있겠으나 유사하게 gaussian 유지



$$g(x) = \begin{bmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_m(x) \end{bmatrix}$$

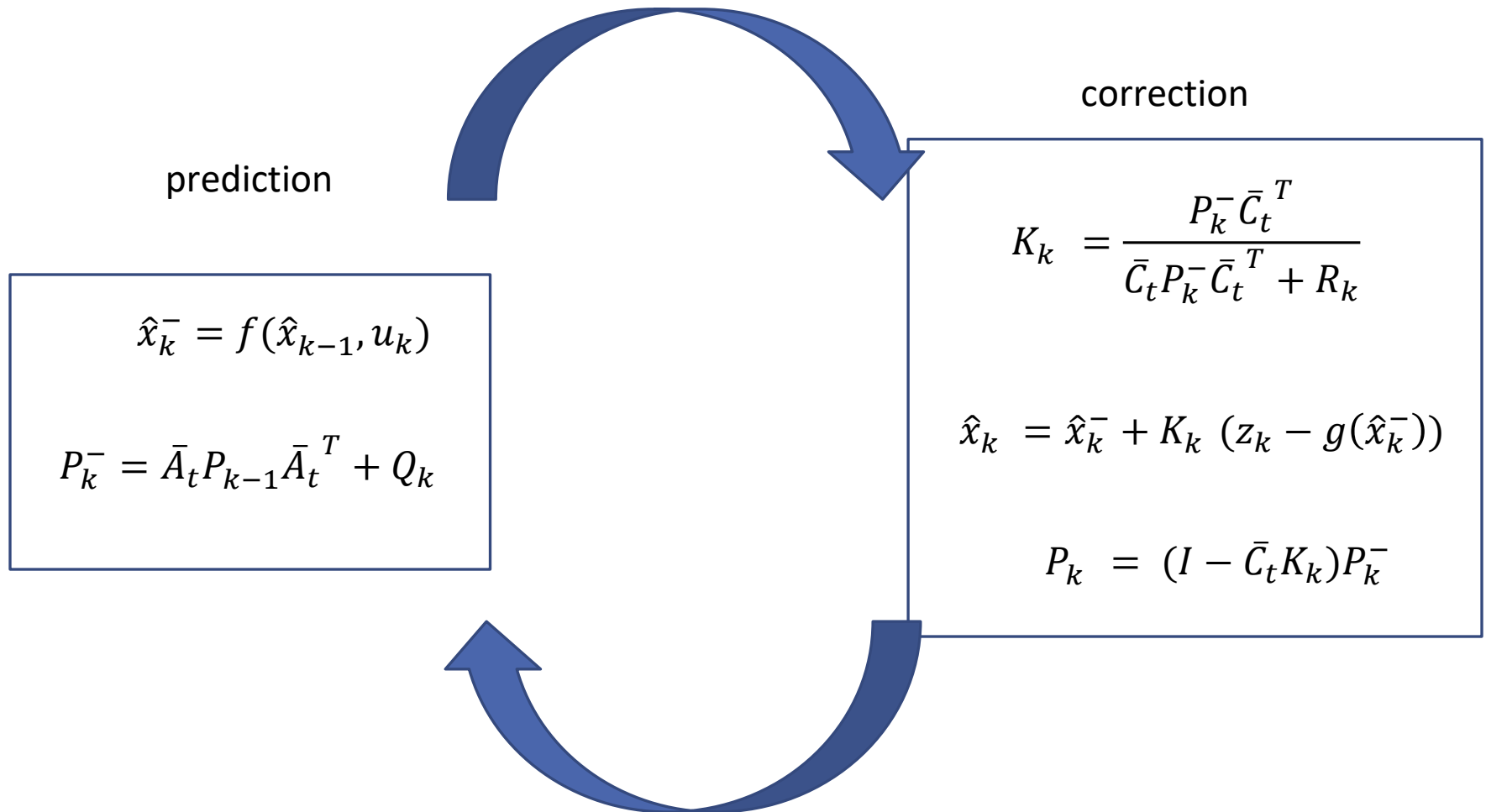
$$G_x = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \dots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \dots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial g_m}{\partial x_1} & \frac{\partial g_m}{\partial x_2} & \dots & \frac{\partial g_m}{\partial x_n} \end{bmatrix}$$

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

Estimation

➤ Extended Kalman Filter

- 칼만필터와 전체적인 구조는 같다. 예측의 평균은 비선형 모델 그대로 쓴다.



Estimation

➤ 예측(prediction)

- 이전 단계의 추정값($\hat{x}_{(k-1)}$)과 입력 (u_k) 을 가지고 상태 예측(\hat{x}_k^-, P_k^-).
- 평균 예측 \hat{x}_k^- 은 비선형 모델 f 를 그대로 쓴다.
- f 에 대해서 자코비안 행렬로 선형화

prediction

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_k)$$

$$P_k^- = \bar{A}_t P_{k-1} \bar{A}_t^T + Q_k$$

$$x_{k+1} = f(x_k, u_k) + w_k$$

$$\bar{A}_t = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

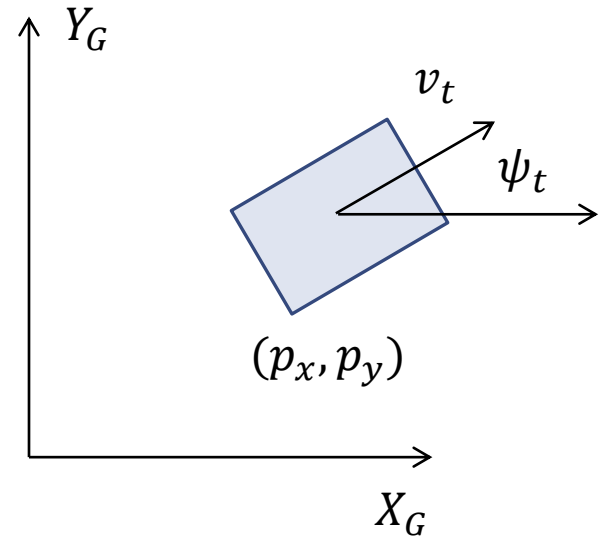
Estimation

예측(prediction)

- 이전 단계의 추정값($\hat{x}_{(k-1)}$)과 입력(u_k)을 가지고 상태 예측(\hat{x}_k, P_k).
- 평균 예측 \hat{x}_k 은 비선형 모델 f 를 그대로 쓴다.
- f 에 대해서 자코비안 행렬로 선형화

(e.g)

$$\dot{X} = \frac{d}{dt} \begin{bmatrix} p_x \\ p_y \\ \psi_t \end{bmatrix}$$
$$= f(X, u) = \begin{bmatrix} u_1 \cos(\psi_t) \\ u_1 \sin(\psi_t) \\ u_2 \end{bmatrix}$$



Estimation

예측(prediction)

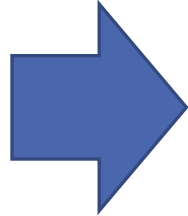
- 이전 단계의 추정값($\hat{x}_{(k-1)}$)과 입력 (u_k) 을 가지고 상태 예측(\hat{x}_k, P_k).
- 평균 예측 \hat{x}_k 은 비선형 모델 f 를 그대로 쓴다.
- f 에 대해서 자코비안 행렬로 선형화

$$f_c(X, u) = \begin{bmatrix} u_1 \cos(\psi_t) \\ u_1 \sin(\psi_t) \\ u_2 \end{bmatrix} \quad \Rightarrow \quad \frac{df_c}{dt} = \frac{\partial f_c}{\partial X} \frac{dX}{dt}$$

$$\Rightarrow \quad \frac{\partial f_c}{\partial X} = \begin{bmatrix} 0 & 0 & -u_1 \sin(\psi_t) \\ 0 & 0 & u_1 \cos(\psi_t) \\ 0 & 0 & 0 \end{bmatrix}$$

➤ 예측(prediction)

- 이전 단계의 추정값($\hat{x}_{(k-1)}$)과 입력 (u_k) 을 가지고 상태 예측(\hat{x}_k, P_k).
- 평균 예측 \hat{x}_k 은 비선형 모델 f 를 그대로 쓴다.
- f 에 대해서 자코비안 행렬로 선형화


$$\frac{\partial f_c}{\partial U} = \begin{bmatrix} \cos(\psi_t) & 0 \\ \sin(\psi_t) & 0 \\ 0 & 1 \end{bmatrix}$$

➤ 예측(prediction)

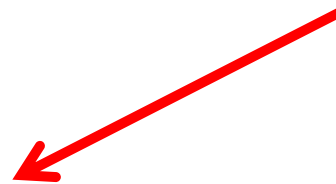
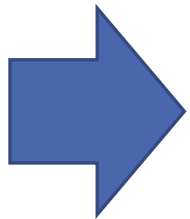
- 이전 단계의 추정값($\hat{x}_{(k-1)}$)과 입력 (u_k) 을 가지고 상태 예측(\hat{x}_k^-, P_k^-).
- 평균 예측 \hat{x}_k^- 은 비선형 모델 f 를 그대로 쓴다.
- f 에 대해서 자코비안 행렬로 선형화

$$\begin{aligned} X_{t+1} &= f(X_k, u_k) \cong X_t + f_c \Delta t \cong X_t + \Delta t \left(\frac{\partial f_c}{\partial X} X_t + \frac{\partial f_c}{\partial u} u_t \right) \\ &= \left(I + \Delta t \frac{\partial f}{\partial X} \bigg|_{X=X_t} \right) X_t + \Delta t \frac{\partial f_c}{\partial u} \bigg|_{u=u_t} u = \bar{A}_t X_t + \bar{B}_t u_t \end{aligned}$$

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_k)$$

$$P_k^- = \bar{A}_t P_{k-1} \bar{A}_t^T + Q_k$$

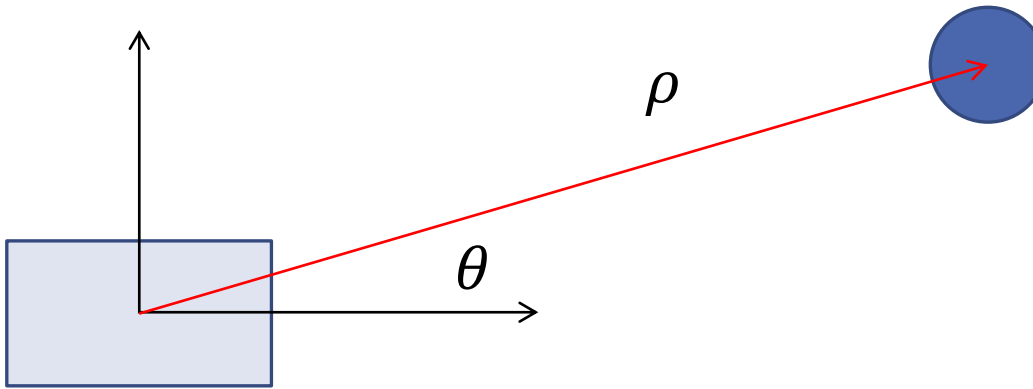
본래 공분산보다 부정확한 건
어느 정도 감수해야.



Estimation

➤ 보정(correction)

- 상태 예측(\hat{x}_{k^-}, P_{k^-})한 결과에 측정값(z_k)을 반영해서 보정하고 상태 추정 (\hat{x}_k, P_k) 확정.
- 측정 모델은 그대로 쓴다.
- 행렬 C 는 측정 모델 $g(x)$ 를 선형화.

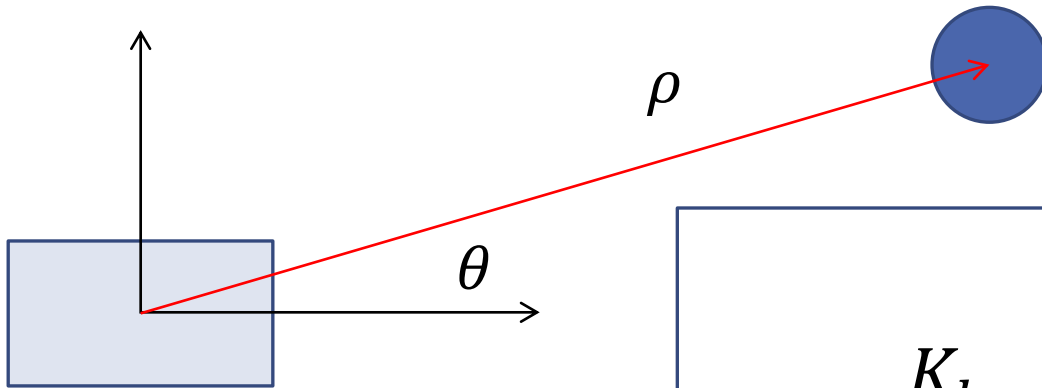


$$X = \begin{bmatrix} p_x \\ p_y \end{bmatrix} \quad \Rightarrow \quad Z = \begin{bmatrix} \rho \\ \theta \end{bmatrix} = g(X) \cong C \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$

Estimation

➤ 보정(correction)

- 상태 예측(\hat{x}_{k^-}, P_{k^-})한 결과에 측정값(z_k)을 반영해서 보정하고 상태 추정 (\hat{x}_k, P_k) 확정.
- 측정 모델은 그대로 쓴다.
- 행렬 C 는 측정 모델 $g(x)$ 를 선형화.



$$K_k = \frac{P_k^- \bar{C}_t^T}{\bar{C}_t P_k^- \bar{C}_t^T + R_k}$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - g(\hat{x}_k^-))$$

$$P_k = (I - \bar{C}_t K_k) P_k^-$$

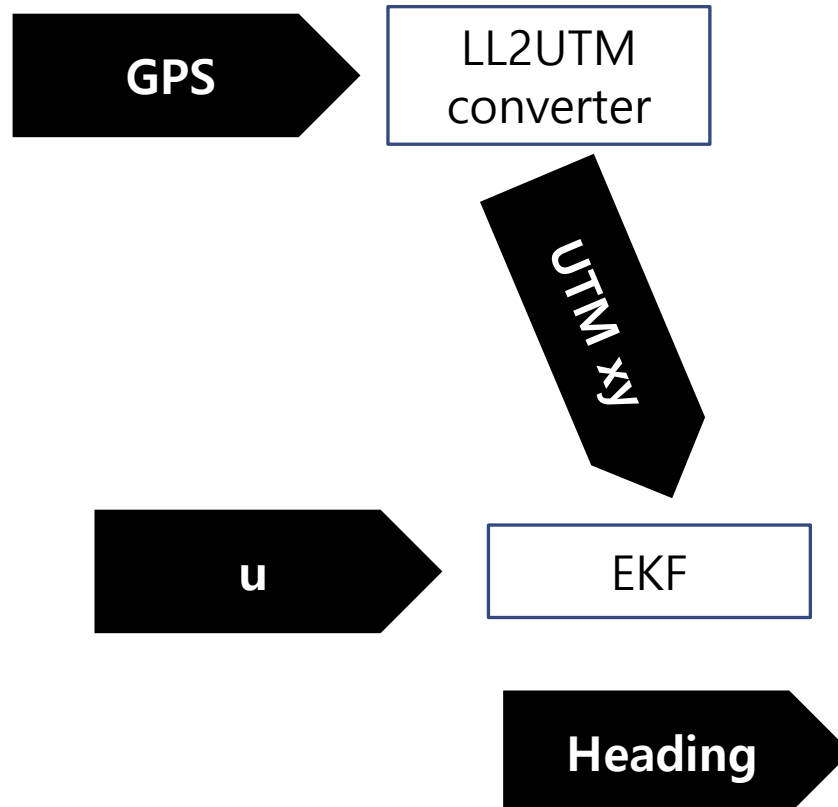
Estimation

Kalman Filter 실습

Estimation

➤ Heading estimation

- 전체 프로세스



Estimation

➤ Heading estimation

- ekf_estimator 노드 작성
 - ✓ kalman filter

```
class ExtendedKalmanFilter:
    def __init__(self, dt=0.05):

        self.T = dt

        self.X = np.array([0,0,0], dtype=float).reshape([-1,1])

        self.P = np.diag([2,2,2])

        self.Q = self.T*np.diag([0.1, 0.1, 0.02])

        self.R = np.diag([0.1, 0.1])/self.T

        self.H = np.array(
            [
                [1,0,0],
                [0,1,0]
            ],
            dtype=float)

        self.pose_pub = rospy.Publisher('/pose_ekf', Odometry, queue_size=1)

        self.odom_msg=Odometry()
        self.odom_msg.header.frame_id='/map'
```

Estimation

➤ Heading estimation

- ekf_estimator 노드 작성
 - ✓ kalman filter

$$\begin{aligned}\hat{x}_k^- \\ &= f(\hat{x}_{k-1}, u_k) \\ P_k^- \\ &= \bar{A}_t P_{k-1} \bar{A}_t^T + Q_k\end{aligned}$$

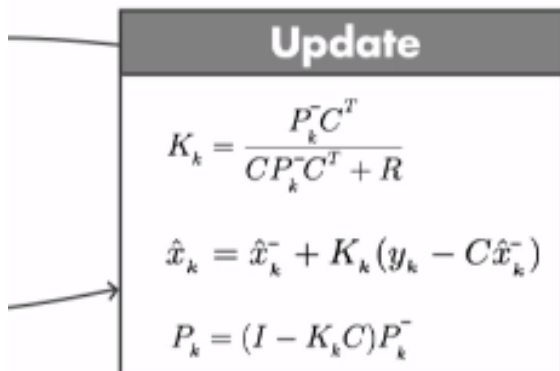
```
def prediction(self, u):  
  
    dX_pre = np.zeros((3, 1), dtype=float)  
    dX_pre[0, :] = u[0]*np.cos(self.X[2, :])  
    dX_pre[1, :] = u[0]*np.sin(self.X[2, :])  
    dX_pre[2, :] = u[1]  
  
    self.X+=(self.T*dX_pre)  
  
    if self.X[2, :] < -np.pi:  
        self.X[2, :] = self.X[2, :] + 2*np.pi  
  
    elif self.X[2, :] > np.pi:  
        self.X[2, :] = self.X[2, :] - 2*np.pi  
  
    else:  
        pass  
  
    self.calc_F(self.X, u)  
  
    self.P = self.F.dot(self.P).dot(self.F.T) + self.Q
```

Estimation

➤ Heading estimation

- ekf_estimator 노드 작성
 - ✓ kalman filter

```
def correction(self, Z):  
    K = self.P.dot(self.H.T).dot(np.linalg.inv(self.H.dot(self.P).dot(self.H.T) + self.R))  
    Y = self.H.dot(self.X)  
  
    self.X = self.X + K.dot(Z-Y)  
  
    self.P = self.P - K.dot(self.H).dot(self.P)  
  
def calc_F(self, x, u):  
    self.F = np.zeros_like(self.Q, dtype=float)  
  
    self.F[0,2] = -u[0]*np.sin(x[2, :])  
    self.F[1,2] = u[0]*np.cos(x[2, :])  
  
    self.F = np.identity(x.shape[0]) + self.T*self.F
```

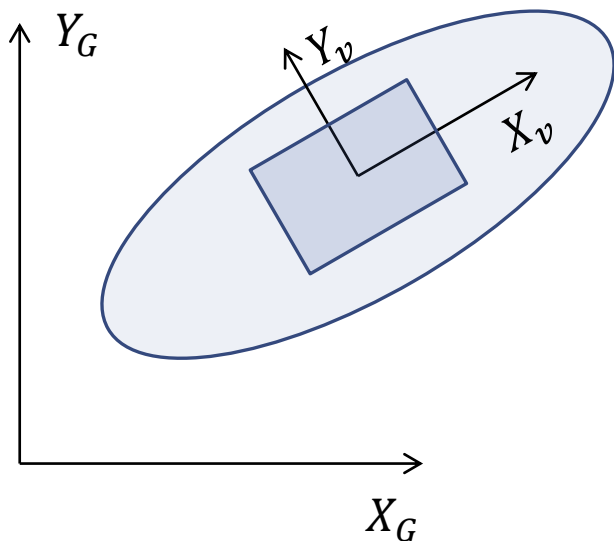


Estimation

➤ Heading estimation

- ekf_estimator 노드 작성
 - ✓ Kalman filter estimated pose publish
 - ✓ Odometry 메시지로 작성
 - ✓ Covariance : 6X6 행렬이 디폴트, (x,y,z,roll,pitch,yaw)
 - ✓ 현재 모델링된 P는 3X3 행렬. 확장 필요

$$P = \begin{bmatrix} P_{xx} & P_{xy} & P_{x\psi} \\ P_{yx} & P_{yy} & P_{y\psi} \\ P_{\psi x} & P_{\psi y} & P_{\psi\psi} \end{bmatrix}$$



$$P = \begin{bmatrix} P_{xx} & P_{xy} & 0 & 0 & 0 & P_{x\psi} \\ P_{yx} & P_{yy} & 0 & 0 & 0 & P_{y\psi} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ P_{\psi x} & P_{\psi y} & 0 & 0 & 0 & P_{\psi\psi} \end{bmatrix}$$

Estimation

➤ Heading estimation

- ekf_estimator 노드 작성
 - ✓ Kalman filter estimated pose publish

```
def send_estimated_state(self):  
  
    pose_cov_ekf = PoseWithCovariance()  
  
    q = tf.transformations.quaternion_from_euler(0, 0, self.X[2][0])  
  
    pose_cov_ekf.pose.position.x = self.X[0][0]  
    pose_cov_ekf.pose.position.y = self.X[1][0]  
    pose_cov_ekf.pose.position.z = 0.  
  
    pose_cov_ekf.pose.orientation.x = q[0]  
    pose_cov_ekf.pose.orientation.y = q[1]  
    pose_cov_ekf.pose.orientation.z = q[2]  
    pose_cov_ekf.pose.orientation.w = q[3]  
  
    P_3d = np.zeros((6,6))  
    P_3d[:2, :2] = self.P[:2, :2]  
    P_3d[-1, :2] = self.P[-1, :2]  
    P_3d[:2, -1] = self.P[:2, -1]  
    P_3d[-1, -1] = self.P[-1, -1]
```

Estimation

➤ Heading estimation

- ekf_estimator 노드 작성
 - ✓ Kalman filter estimated pose publish

```
P_3d[-1, -1] = self.P[-1, -1]

pose_cov_ekf.covariance = P_3d.reshape([-1]).tolist()

self.odom_msg.pose = pose_cov_ekf

self.pose_pub.publish(self.odom_msg)

br = tf.TransformBroadcaster()

br.sendTransform((self.X[0][0], self.X[1][0], 0.),
                 tf.transformations.quaternion_from_euler(0, 0, self.X[2][0]),
                 rospy.Time.now(),
                 "base_link2",
                 "map")
```

Estimation

➤ Heading estimation

- ekf_estimator 노드 작성
 - ✓ Command 생성기

```
class CMDPublisher:
    def __init__(self):
        self.cmd_pub = rospy.Publisher("/cmd_vel", Twist, queue_size=1)
        self.u = np.zeros((2,))
        self.cmd_msg = Twist()
        self.t = 0.

    def calc_u(self):
        self.u[0] = 0.2
        self.u[1] = 0.5*np.sin(self.t*2*np.pi/5)
        self.cmd_msg.angular.z = self.u[1]
        self.cmd_msg.linear.x = self.u[0]
        self.t+=0.05
        self.cmd_pub.publish(self.cmd_msg)
        return self.u
```

Estimation

➤ Heading estimation

- ekf_estimator 노드 작성
 - ✓ Main loop

```
if __name__ == '__main__':  
    rospy.init_node('EKF_estimator', anonymous=True)  
  
    rate = rospy.Rate(20)  
  
    loc_sensor = Converter()  
  
    ekf = ExtendedKalmanFilter(dt=0.05)  
  
    cmd_gen = CMDPublisher()
```


Estimation

➤ Heading estimation

- ekf_estimator 노드 작성
 - ✓ Main loop

```
cmd_gen = CMDPublisher()

while not rospy.is_shutdown():

    if loc_sensor.x is not None and loc_sensor.y is not None:

        #decide the u
        u = cmd_gen.calc_u()

        #prediction step
        ekf.prediction(u)

        #measurement locations
        z = np.array([loc_sensor.x, loc_sensor.y]).reshape([-1,1])

        #correction step
        ekf.correction(z)

        #get the estimated states
        ekf.send_estimated_state()

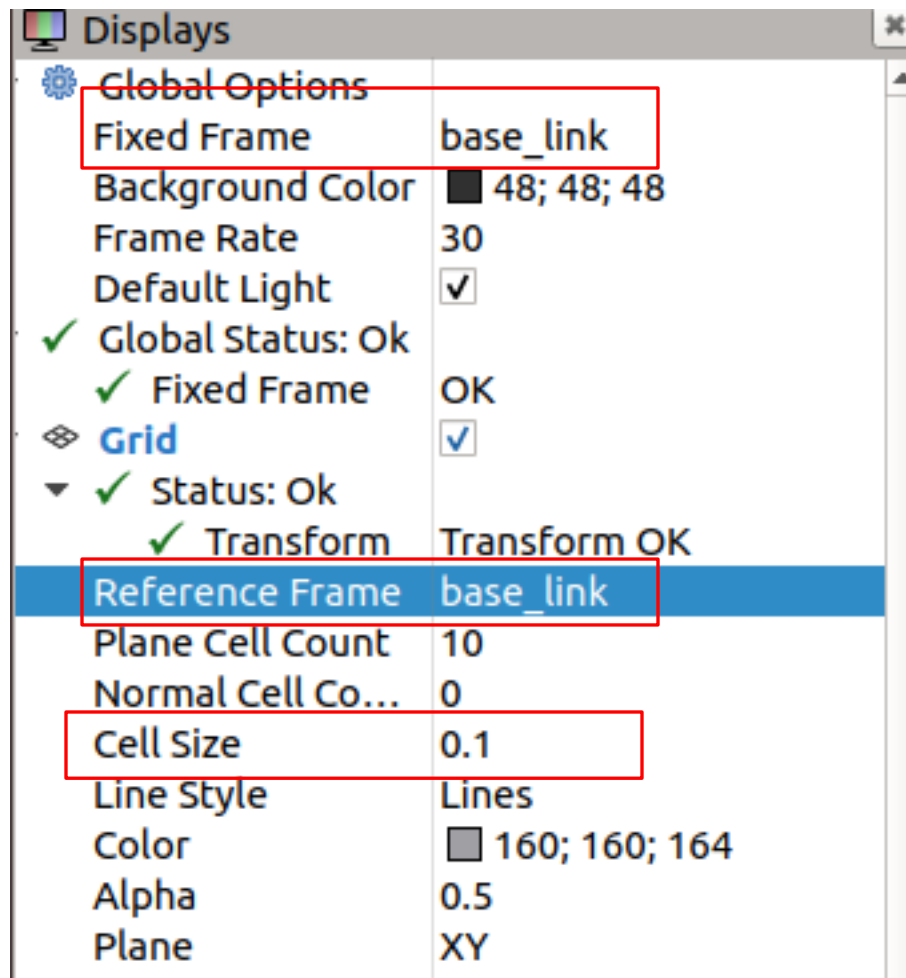
    else:
        pass

    rate.sleep()
```

Estimation

➤ Heading estimation







- ekf_estimator 노드 작성
 - ✓ Rviz 세팅



Estimation

➤ Heading estimation

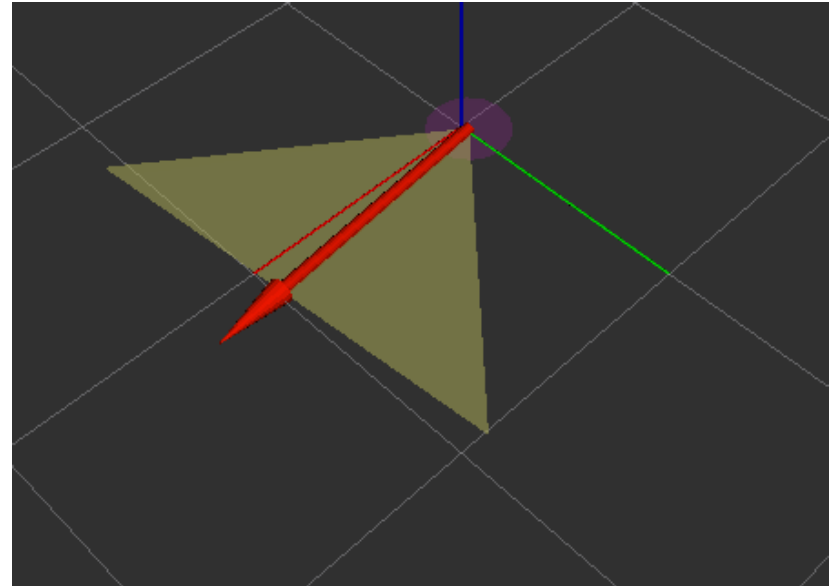
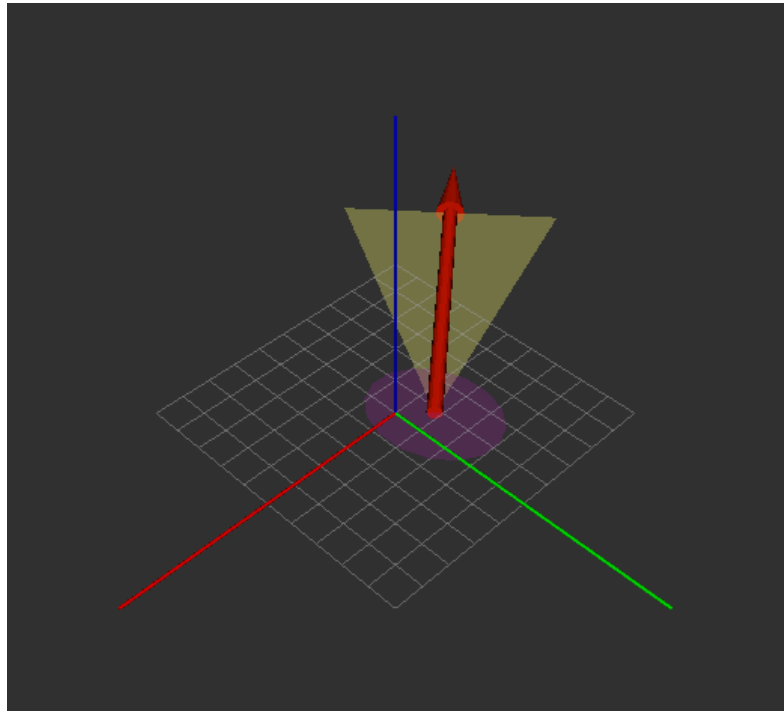
- ekf_estimator 노드 작성
 - ✓ Rviz 세팅

 Odometry	<input checked="" type="checkbox"/>
 Status: Ok	
 Topic	4780 messages receiv...
 Transform...	Transform OK
Topic	/pose_ekf
Unreliable	<input type="checkbox"/>
Position Toler...	0.1
Angle Tolerance	0.1
Keep	1
▶ Shape	Arrow
▶ Covariance	<input checked="" type="checkbox"/>
 Axes	<input checked="" type="checkbox"/>
 Status: Ok	
Reference Frame	base_link
Length	1
Radius	0.01

Estimation

➤ Heading estimation

- ekf_estimator 실행 결과

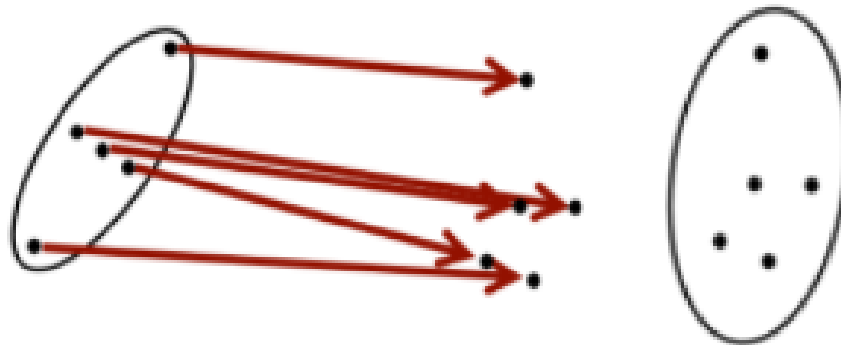


Estimation-UKF

Unscented Kalman Filter

➤ Unscented transform

- 선형화 없이 비선형 모델 출력을 가우시안으로 유지하기 위한 방식
 - 1. sigma point들을 계산한다.
 - 2. 각 sigma point들의 weight를 계산한다.
 - 3. 비선형 함수를 통해 sigma point들의 출력을 계산한다.
 - 4. 출력된 point들의 새로운 Gaussian 분포를 계산한다.
-
- EKF가 사용하는 mean값 근처에서의 선형화를 피하기 위한 방법.



http://jinyongjeong.github.io/2017/02/17/lec06_UKF/

➤ Unscented transform

- sigma point
- N은 상태의 차원. Lambda는 scaling factor
- Sigma point 들이 평균으로부터 멀리 떨어지지 않도록 조정

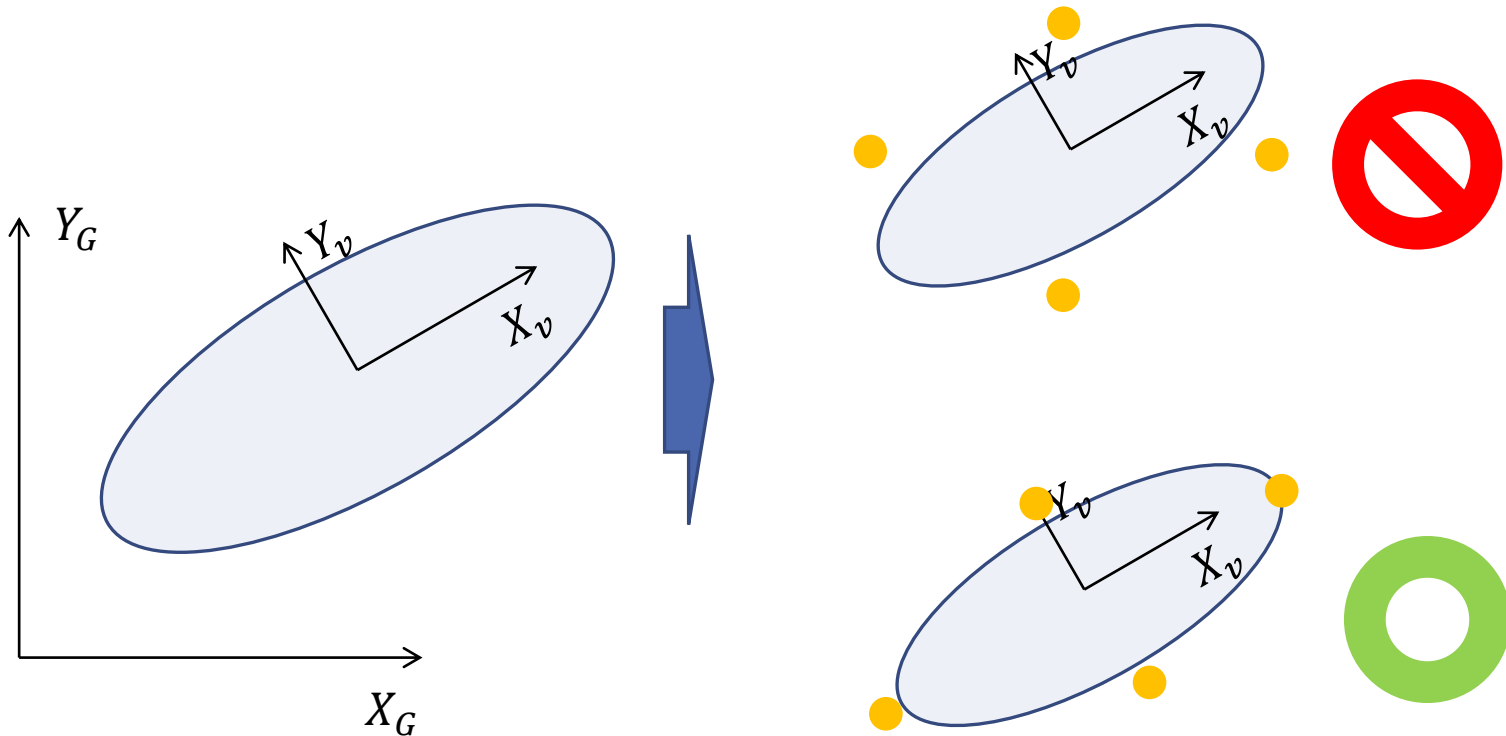
$$\begin{aligned}\chi^{[0]} &= \mu \\ \chi^{[i]} &= \mu + (\sqrt{(n + \lambda)\Sigma})^i \text{ for } i = 1, \dots, n \\ \chi^{[i]} &= \mu - (\sqrt{(n + \lambda)\Sigma})^{i-n} \text{ for } i = n + 1, \dots, 2n\end{aligned}$$

http://jinyongjeong.github.io/2017/02/17/lec06_UKF/

Estimation

➤ Unscented transform

- Eigenvalue decomposition이나 Cholesky decomposition을 사용한 sigma point 생성
- Python에서는 `np.linalg.eiv`, `np.linalg.cholesky()`로 계산 가능



http://jinyongjeong.github.io/2017/02/17/lec06_UKF/

Estimation

➤ Unscented transform

- Sigma point의 가중치 선택
- Sigma point와 가중치를 사용한 estimated x 의 mean, covariance 계산

$$\begin{aligned}\omega_m^{[0]} &= \frac{\lambda}{n + \lambda} & \kappa &\geq 0 \\ \omega_c^{[0]} &= \omega_m^{[0]} + (1 - \alpha^2 + \beta) & \alpha &\in (0, 1] \\ \omega_m^{[i]} &= \omega_c^{[i]} = \frac{1}{2(n + \lambda)} \text{ for } i = 1, \dots, 2n & \lambda &= \alpha^2(n + \kappa) - n \\ & & \beta &= 2\end{aligned}$$

$$\begin{aligned}\mu' &= \sum_{i=0}^{2n} \omega_m^{[i]} g(\chi^{[i]}) \\ \Sigma' &= \sum_{i=0}^{2n} \omega_c^{[i]} (g(\chi^{[i]}) - \mu')(g(\chi^{[i]}) - \mu')^T\end{aligned}$$

http://jinyongjeong.github.io/2017/02/17/lec06_UKF/

Estimation

➤ Unscented transform

- 전체 프로세스

Calculate sigma points:

$$\mathbf{x}_{k-1}^a = [\hat{\mathbf{x}}_{k-1}^a \quad \hat{\mathbf{x}}_{k-1}^a \pm \sqrt{(L + \lambda) \mathbf{P}_{k-1}^a}]$$

Prediction

$$\mathbf{x}_{k|k-1}^x = \mathbf{F}[\mathbf{x}_{k-1}^x, \mathbf{x}_{k-1}^v]$$

$$\hat{\mathbf{x}}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \mathbf{x}_{i,k|k-1}^x$$

$$\mathbf{P}_k^- = \sum_{i=0}^{2L} W_i^{(c)} [\mathbf{x}_{i,k|k-1}^x - \hat{\mathbf{x}}_k^-][\mathbf{x}_{i,k|k-1}^x - \hat{\mathbf{x}}_k^-]^T$$

<https://www.seas.harvard.edu/courses/cs281/papers/unscented.pdf>

Estimation

➤ Unscented transform

- 전체 프로세스

Update

$$\begin{aligned}\tilde{\mathbf{y}}_{k|k-1} &= \tilde{\mathbf{H}}[\tilde{\mathbf{x}}_{k|k-1}^x, \mathbf{x}_{k-1}^m] \\ \hat{\mathbf{y}}_k^- &= \sum_{i=0}^{2L} W_i^{(m)} \mathbf{y}_{i,k|k-1}\end{aligned}$$

$$\mathbf{P}_{\hat{\mathbf{y}}_k \hat{\mathbf{y}}_k} = \sum_{i=0}^{2L} W_i^{(c)} [\mathbf{y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-][\mathbf{y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-]^T$$

$$\mathbf{P}_{\mathbf{x}_k \mathbf{y}_k} = \sum_{i=0}^{2L} W_i^{(c)} [\mathbf{x}_{i,k|k-1} - \hat{\mathbf{x}}_k^-][\mathbf{y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-]^T$$

$$\mathcal{K} = \mathbf{P}_{\mathbf{x}_k \mathbf{y}_k} \mathbf{P}_{\hat{\mathbf{y}}_k \hat{\mathbf{y}}_k}^{-1}$$

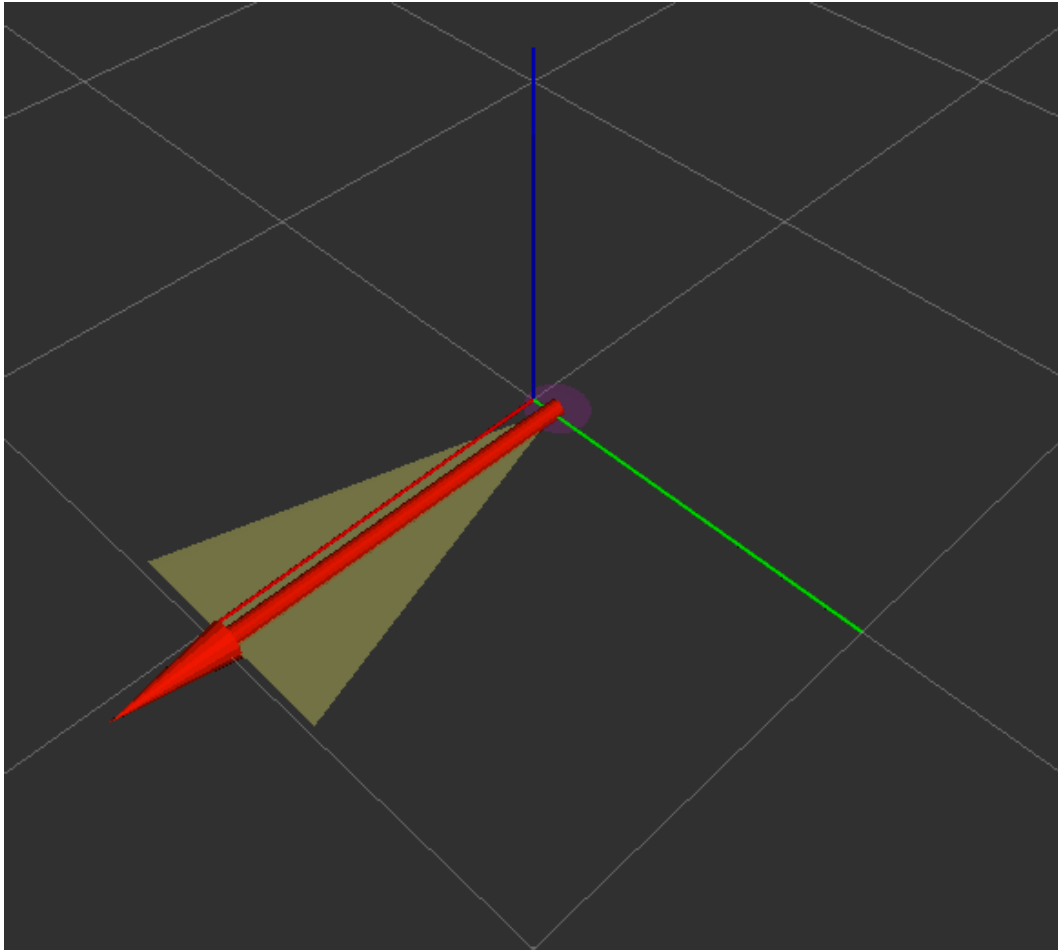
$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathcal{K}(\mathbf{y}_k - \hat{\mathbf{y}}_k^-)$$

$$\mathbf{P}_k = \mathbf{P}_k^- - \mathcal{K} \mathbf{P}_{\hat{\mathbf{y}}_k \hat{\mathbf{y}}_k} \mathcal{K}^T$$

<https://www.seas.harvard.edu/courses/cs281/papers/unscented.pdf>

Estimation

- Unscented transform
 - 실행 결과



<https://www.seas.harvard.edu/courses/cs281/papers/unscented.pdf>

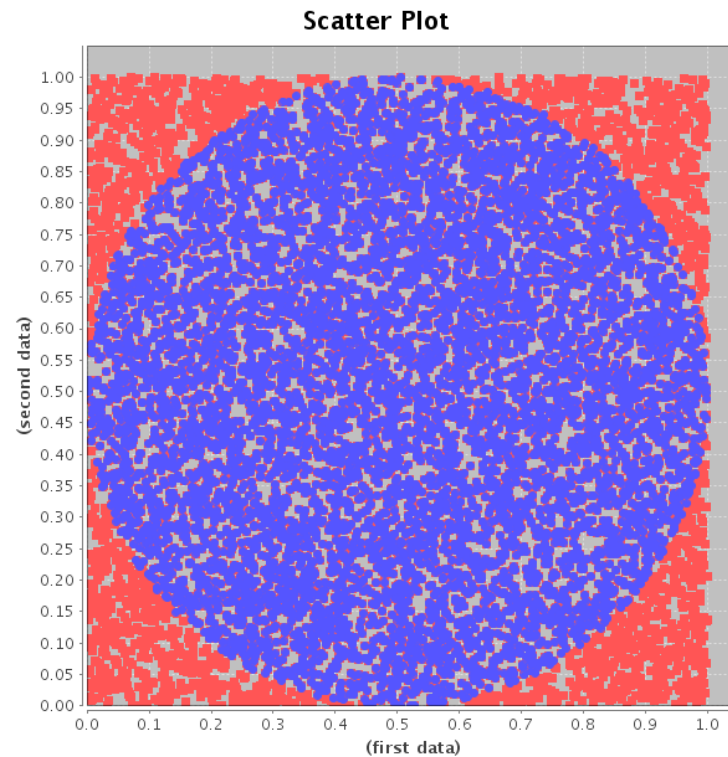
Estimation

Particle Filter based Localization

Estimation

➤ Monte carlo simulation

- 해석적으로 솔루션을 구할 수 없을 때 유용
- 정해진 분포 내에서 가능한 한 많은 샘플을 생성
- 샘플 하나하나의 결과를 취합 후 근사값 도출



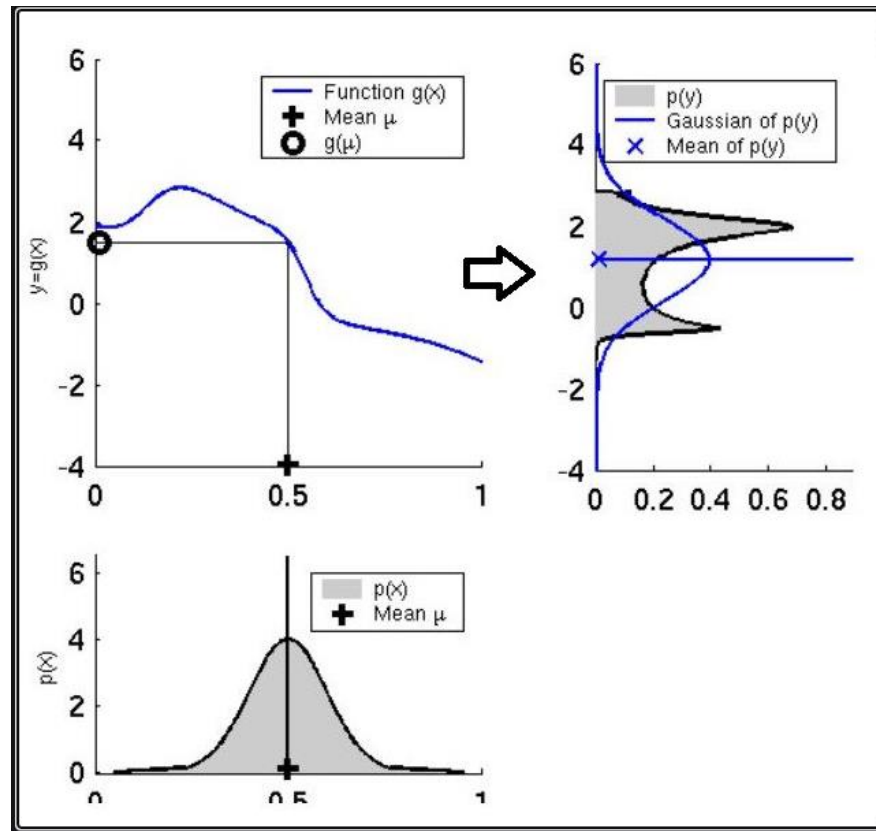
<https://www.r-bloggers.com/2012/10/estimation-of-the-number-pi-a-monte-carlo-simulation/>

Estimation

➤ Particle Filter

- EKF의 한계

- ✓ Non linearity가 심한 시스템에서 예측이 맞지 않음.
- ✓ 노이즈가 non gaussian이면 예측/보정이 맞지 않음.

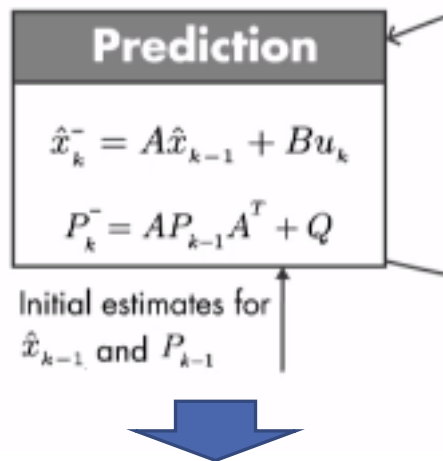


<https://towardsdatascience.com/extended-kalman-filter-43e52b16757d>

Estimation

➤ Particle Filter

- Particle filter의 개념
 - ✓ 1부터 100까지 다 넣어보자.



$$\begin{bmatrix} p \\ v \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ v \end{bmatrix}_k + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_k + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} w_k$$

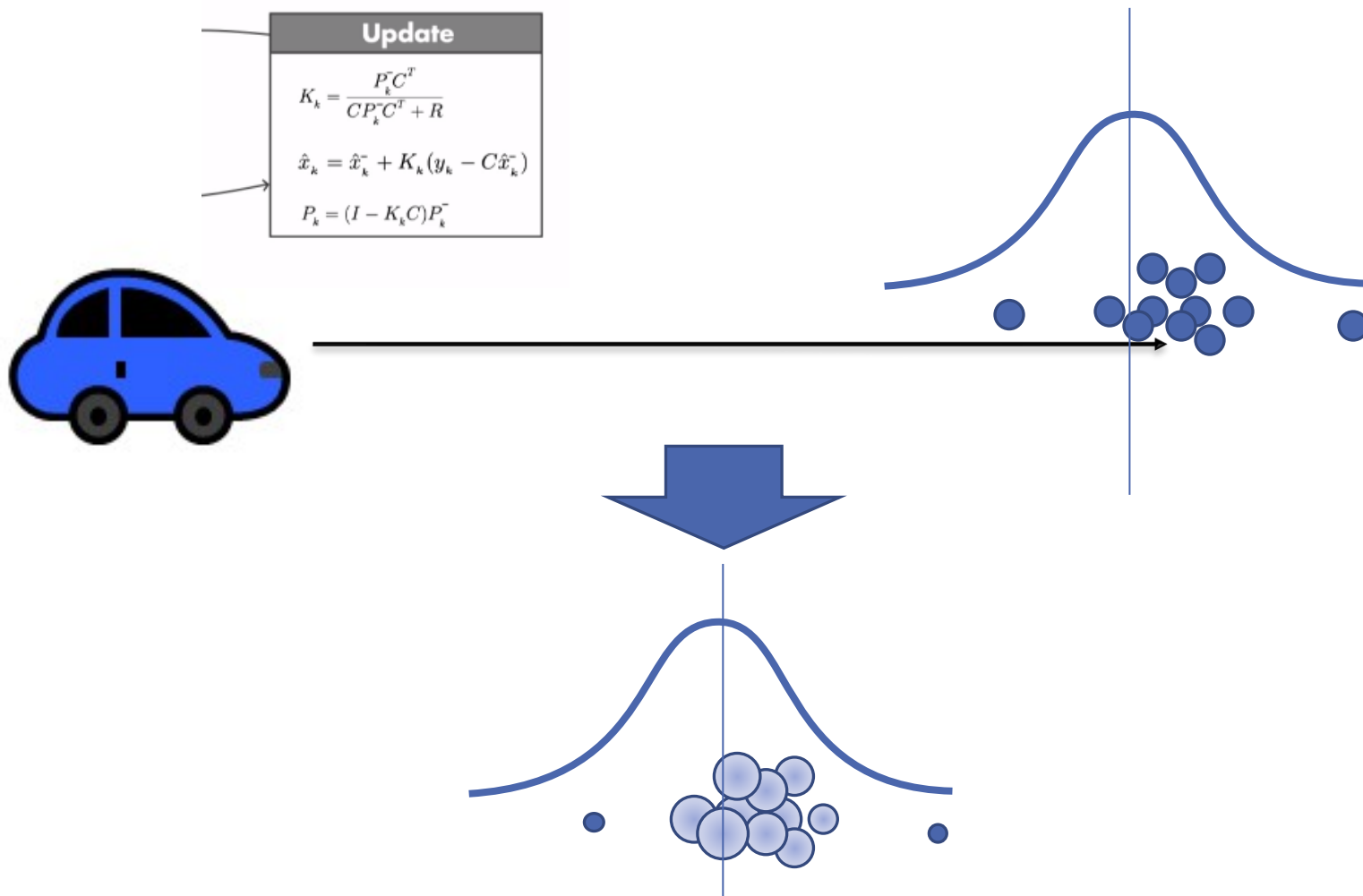
w_k 를 n 번 만큼 샘플링

Estimation

➤ Particle Filter

- Particle filter의 개념

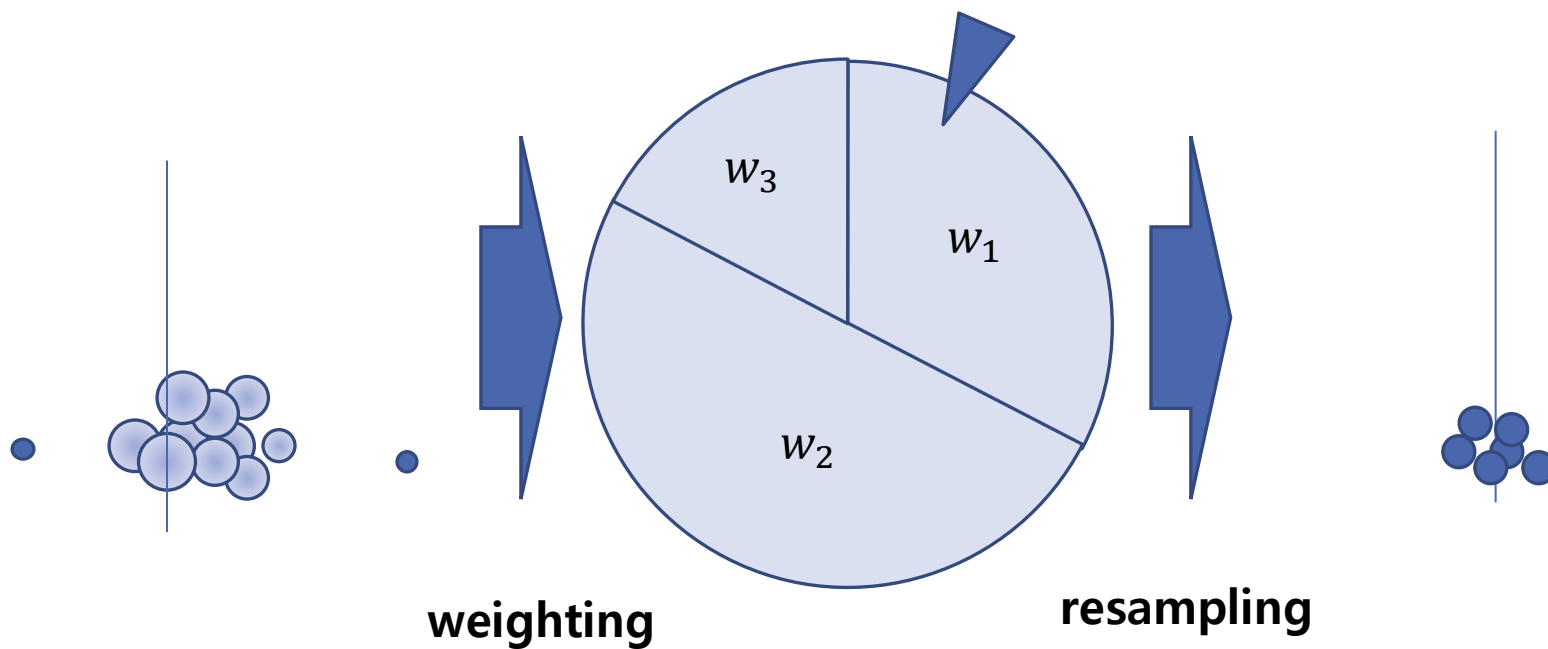
- ✓ Measurement에서 제일 가까운 파티클에 높은 가중치를 주자.



Estimation

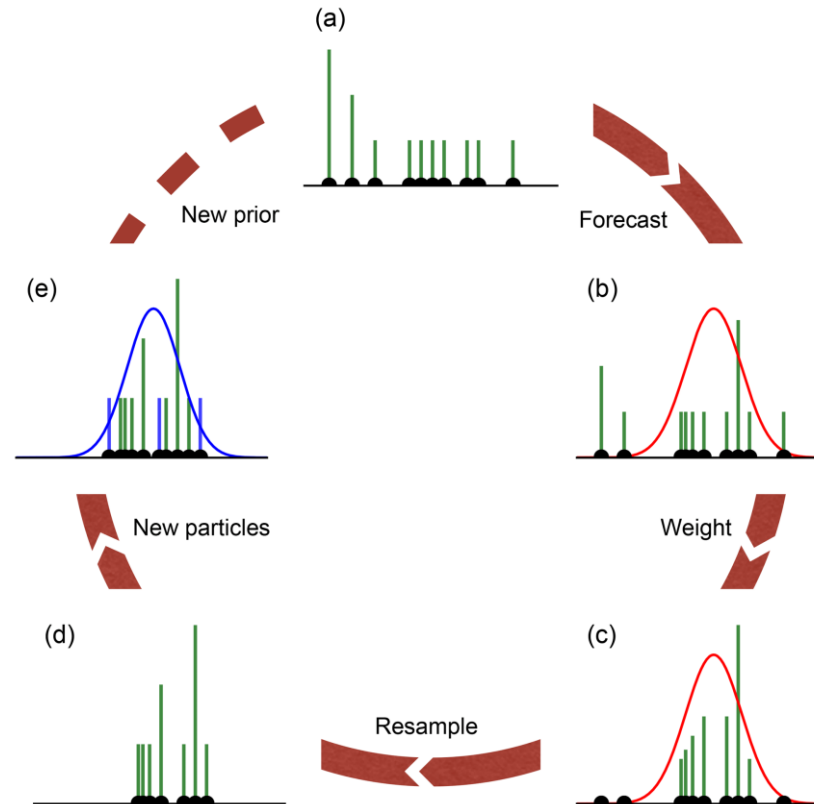
➤ Particle Filter

- Particle filter의 개념
 - ✓ 가중치로 돌림판을 만들어서 뽑기를 하자.



➤ Particle Filter

- Particle filter의 개념
 - ✓ 반복하다보면 실제 차량의 상태대로 추정될 것이다.

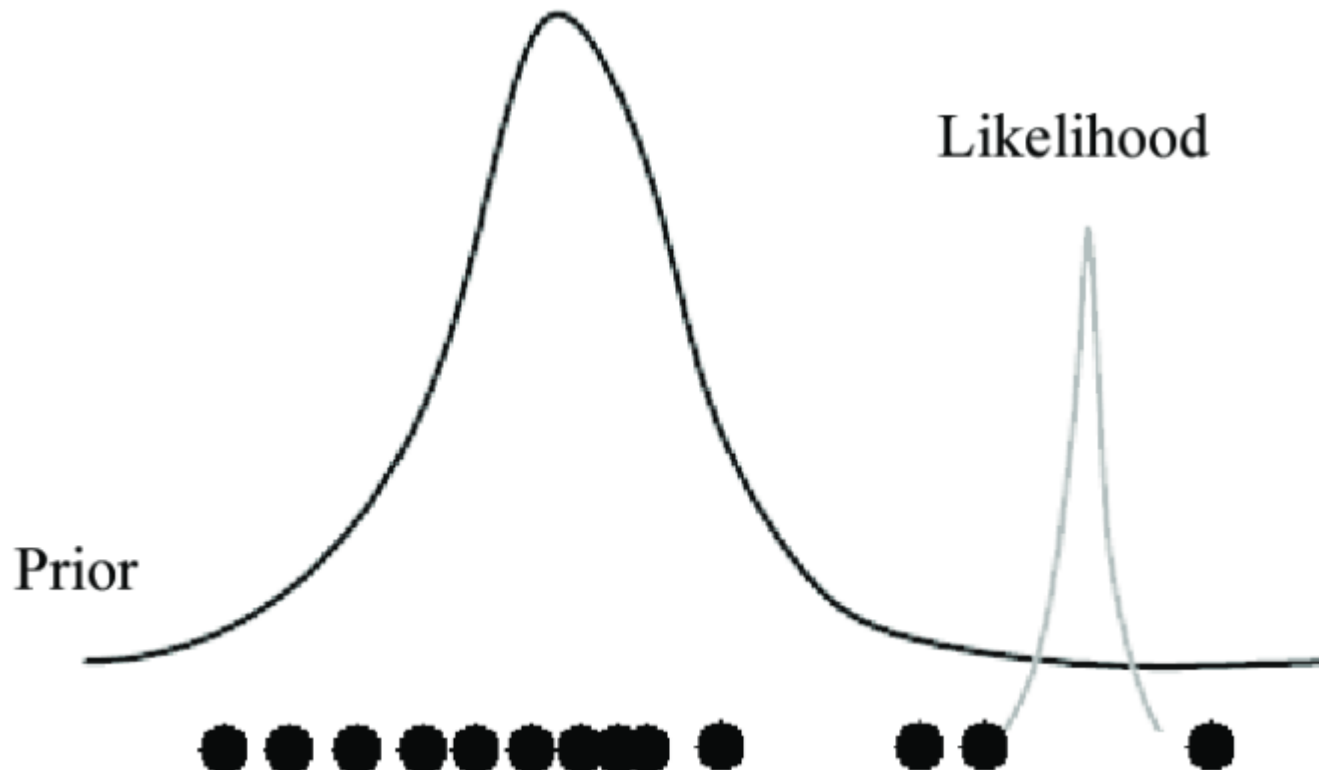


[https://hess.copernicus.org/articles/23/1163/2019/#&gid=1
&pid=1](https://hess.copernicus.org/articles/23/1163/2019/#&gid=1&pid=1)

Estimation

➤ Particle Filter

- Particle filter 단점
 - ✓ Vanishing sample 문제



https://www.researchgate.net/figure/Limitations-in-generic-particle-filter_fig2_225549514

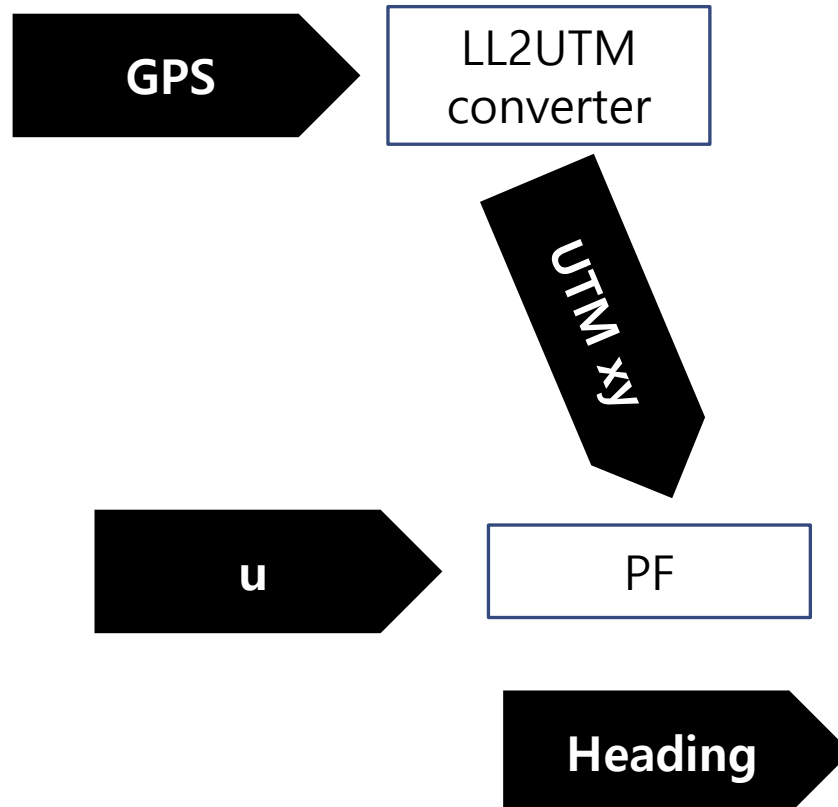
Estimation

Particle Filter 실습

Estimation

➤ Heading estimation with PF

- 전체 프로세스



Estimation

➤ Heading estimation

- Pf_estimator 노드 작성
 - ✓ Multivariate normal distribution pdf

```
def pdf_multivariate_gauss(dx, cov):  
    part2 = np.exp(-0.5*np.matmul(np.matmul(dx.T, np.linalg.inv(cov)), dx))  
    part1 = 1 / ((2* np.pi)**(cov.shape[0]/2) * (np.linalg.det(cov)**(1/2)))  
    return np.diag(part1 * part2) + 0.00001
```

Estimation

➤ Heading estimation

- Pf_estimator 노드 작성
 - ✓ Particle filter

```
class ParticleFilter:
    def __init__(self, dt=0.05, NP=1000):

        self.T = dt

        self.NP = NP

        self.XP = np.random.randn(3, self.NP)*50 + np.array([20, 1100., 0]).reshape([3, 1])

        self.limit_yaw()

        self.pw = np.ones((NP, ))/NP

        self.q = [0.01, 0.01, 0.02]

        self.R = np.diag([0.01, 0.01])/self.T

        self.H = np.array(
            [
                [1,0,0],
                [0,1,0]
            ],
            dtype=float)

        self.pose_pub = rospy.Publisher('/pose_pf', PoseArray, queue_size=1)
```


Estimation

➤ Heading estimation

- Pf_estimator 노드 작성
 - ✓ Particle filter

```
def prediction(self, u):  
  
    dX_pre = np.zeros((3, self.NP), dtype=float)  
    dX_pre[0, :] = u[0]*np.cos(self.XP[2, :]) + self.q[0]*np.random.randn(1, self.NP)  
    dX_pre[1, :] = u[0]*np.sin(self.XP[2, :]) + self.q[1]*np.random.randn(1, self.NP)  
    dX_pre[2, :] = u[1] + self.q[2]*np.random.randn(1, self.NP)  
  
    self.XP+=(self.T*dX_pre)  
  
    self.limit_yaw()  
  
def limit_yaw(self):  
  
    over_bool = self.XP[2,:] < -np.pi  
  
    self.XP[2, over_bool] = self.XP[2, over_bool] + 2*np.pi  
  
    under_bool = self.XP[2,:] > np.pi  
  
    self.XP[2, under_bool] = self.XP[2, under_bool] - 2*np.pi
```

Estimation

➤ Heading estimation

- Pf_estimator 노드 작성
 - ✓ Particle filter

```
def correction(self, Z):  
    dz = self.H.dot(self.XP)-Z  
  
    pdf_z = pdf_multivariate_gauss(dz, self.R)  
  
    self.pw = pdf_z/pdf_z.sum()  
  
    self.Xm = np.dot(self.XP, self.pw).reshape([-1, 1])  
  
    self.Xcov = np.dot(  
        (self.XP-self.Xm),  
        np.diag(self.pw)  
    ).dot((self.XP-self.Xm).T)  
  
    re_sample_ID, self.pw = self.re_sampling(self.pw)  
  
    XP_new = self.XP[:, re_sample_ID]  
  
    self.XP = XP_new + np.diag([0.1, 0.1, 0.01]).dot(np.random.randn(3, self.NP))  
  
    self.limit_yaw()
```

Estimation

➤ Heading estimation

- Pf_estimator 노드 작성
 - ✓ Particle filter

```
def re_sampling(self, w):  
    re_sample_ID = np.random.choice(np.arange(self.NP), self.NP, p=w)  
  
    w = np.ones((self.NP, ))/self.NP  
  
    return re_sample_ID, w
```

Estimation

➤ Heading estimation

- Pf_estimator 노드 작성
 - ✓ Particle filter

```
def send_estimated_state(self):  
  
    particles_msg = PoseArray()  
    particles_msg.header.frame_id='/map'  
  
    for p_i in range(self.NP):  
  
        particle_msg = Pose()  
  
        px = self.XP[:, p_i]  
  
        q = tf.transformations.quaternion_from_euler(0, 0, px[2])  
  
        particle_msg.position.x = px[0]  
        particle_msg.position.y = px[1]  
        particle_msg.position.z = 0.  
  
        particle_msg.orientation.x = q[0]  
        particle_msg.orientation.y = q[1]  
        particle_msg.orientation.z = q[2]  
        particle_msg.orientation.w = q[3]  
  
        particles_msg.poses.append(particle_msg)  
  
    self.pose_pub.publish(particles_msg)
```

Estimation

➤ Heading estimation

- Pf_estimator 노드 작성
 - ✓ Particle filter

```
pose_cov_pf.covariance = P_3d.reshape([-1]).tolist()

self.odom_msg.pose = pose_cov_pf

self.pose_pub.publish(self.odom_msg)
```

Estimation

➤ Heading estimation

- Pf_estimator 노드 작성
 - ✓ Particle filter

```
if __name__ == '__main__':  
    rospy.init_node('PF_estimator', anonymous=True)  
  
    rate = rospy.Rate(20)  
  
    loc_sensor = Converter()  
  
    pf = ParticleFilter(dt=0.05)  
  
    cmd_gen = CMDPublisher()
```

Estimation

➤ Heading estimation

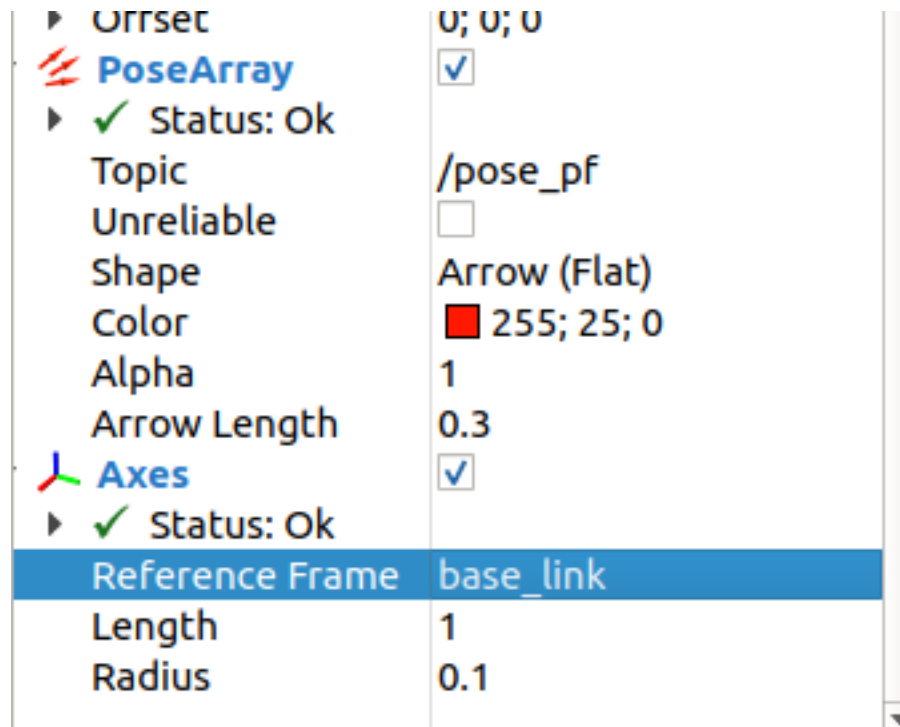
- Pf_estimator 노드 작성
 - ✓ Particle filter

```
while not rospy.is_shutdown():  
    if loc_sensor.x is not None and loc_sensor.y is not None:  
        #decide the u  
        u = cmd_gen.calc_u()  
  
        #prediction step  
        pf.prediction(u)  
  
        #measurement locations  
        z = np.array([loc_sensor.x, loc_sensor.y]).reshape([-1,1])  
  
        #correction step  
        pf.correction(z)  
  
        #get the estimated states  
        pf.send_estimated_state()  
  
    else:  
        pass  
  
    rate.sleep()
```

Estimation

➤ Heading estimation

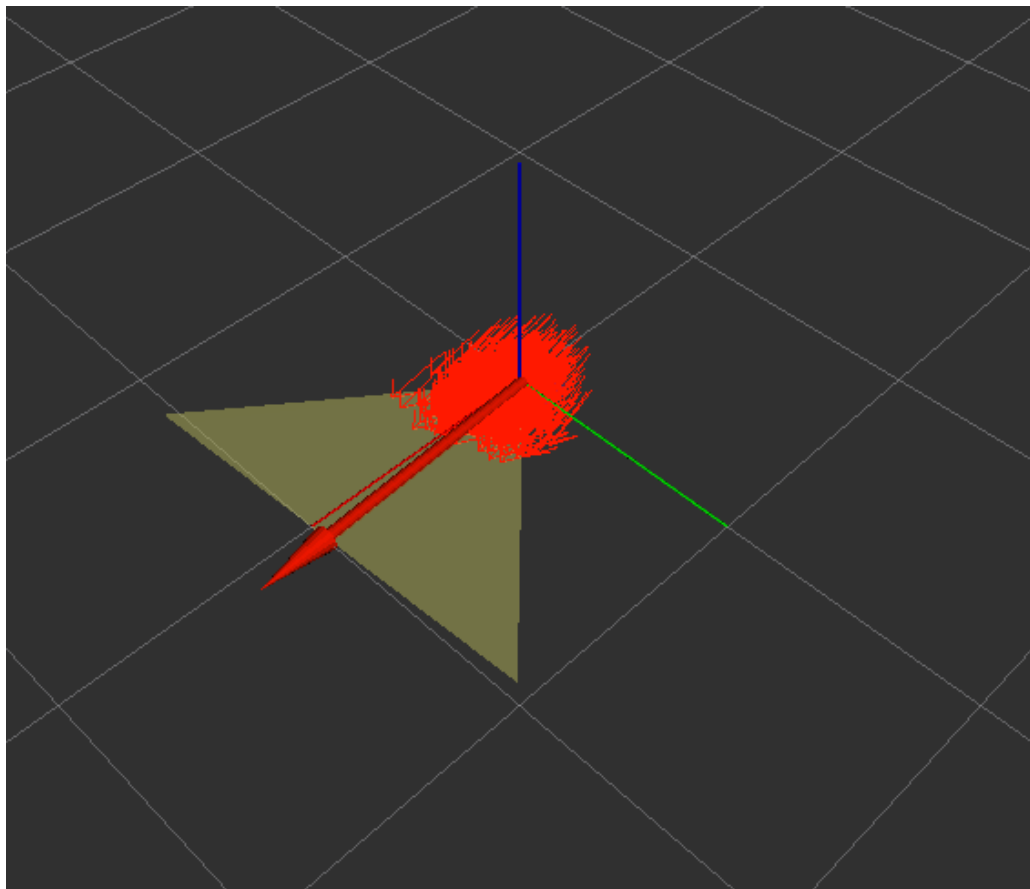
- Rviz 세팅
 - ✓ Particle filter



Estimation

➤ Heading estimation

- Pf_estimator 실행 결과
 - ✓ heading에 대한 covariance는 작게 표현되지만, 딜레이가 존재한다.



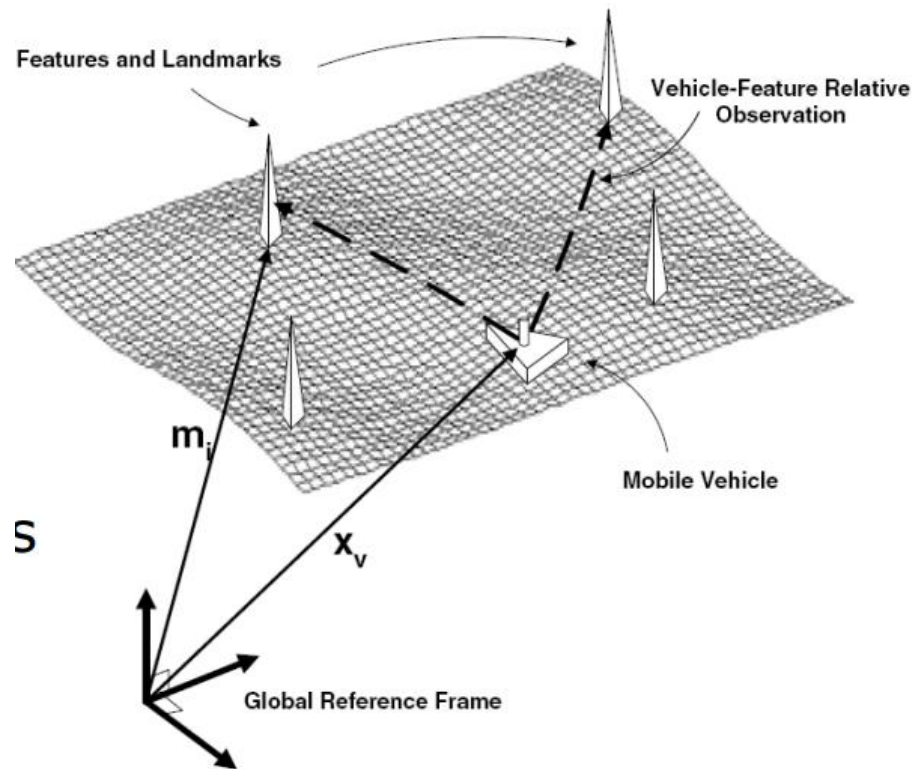
Localization

Particle Filter based Localization

Localization

➤ Localization & Mapping

- Localization : map이 주어지고, 센서 및 feature 알고리즘으로 맵 내 위치 추정
- Mapping : 센서, odometry, 랜드마크 feature 정보를 가지고 맵 생성
- Simultaneous localization and mapping : 맵을 생성해가면서 localization

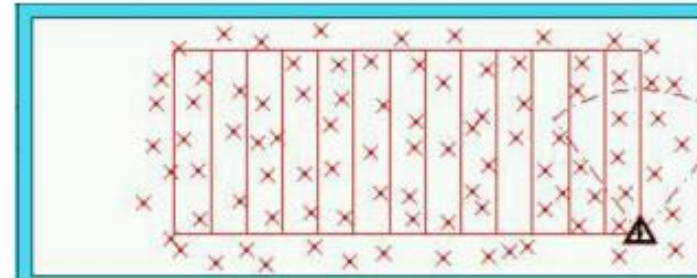
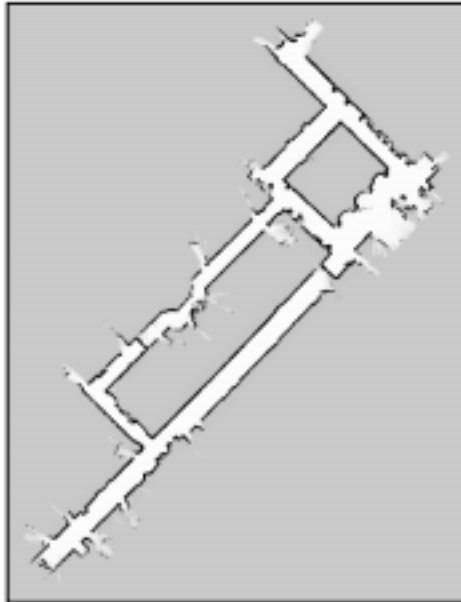


<http://ais.informatik.uni-freiburg.de/teaching/ss12/robotics/slides/12-slam.pdf>

Localization

➤ Localization & Mapping

- Map 의 종류
 - ✓ Grid map
 - ✓ Landmark

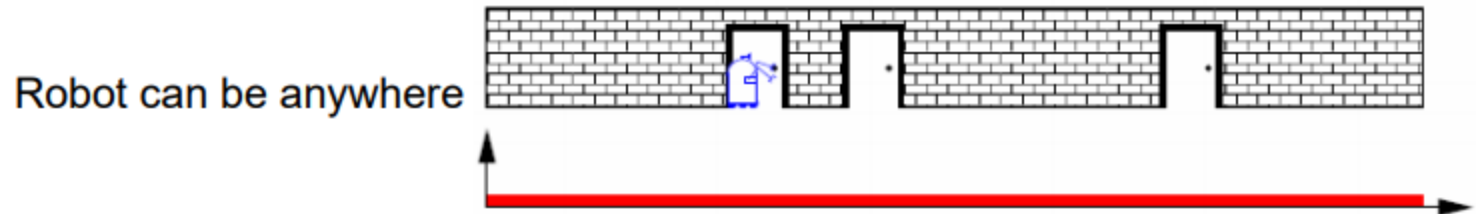


<http://ais.informatik.uni-freiburg.de/teaching/ss12/robotics/slides/12-slam.pdf>

Localization

➤ Localization & Mapping

- 1D-localization example
 - ✓ Land mark : doors
 - ✓ Given the map as a list of points of doors
 - ✓ Initialize the position of a robot anywhere

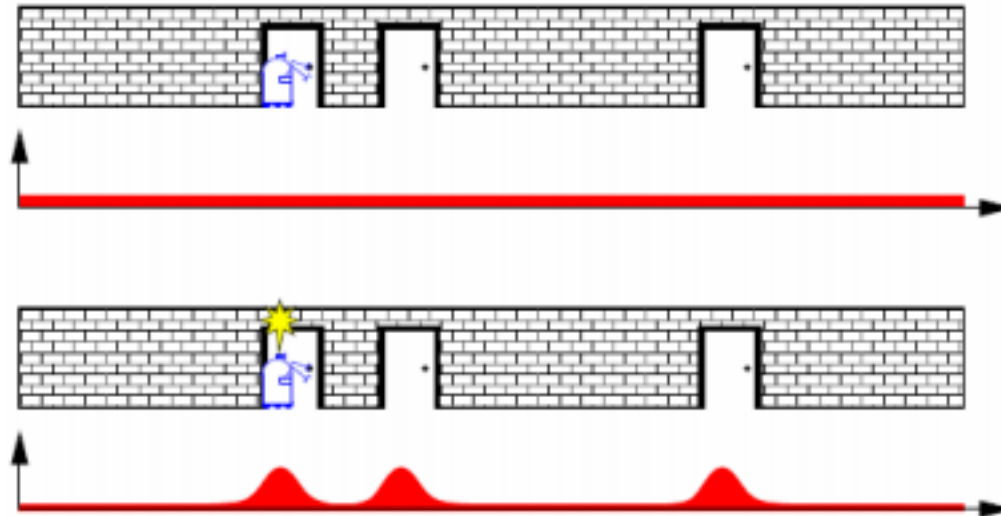


https://www.cs.cmu.edu/~motionplanning/lecture/Chap9-Bayesian-Mapping_howie.pdf

Localization

➤ Localization & Mapping

- 1D-localization example
 - ✓ Detect the land mark
 - ✓ Estimate the distribution of positions

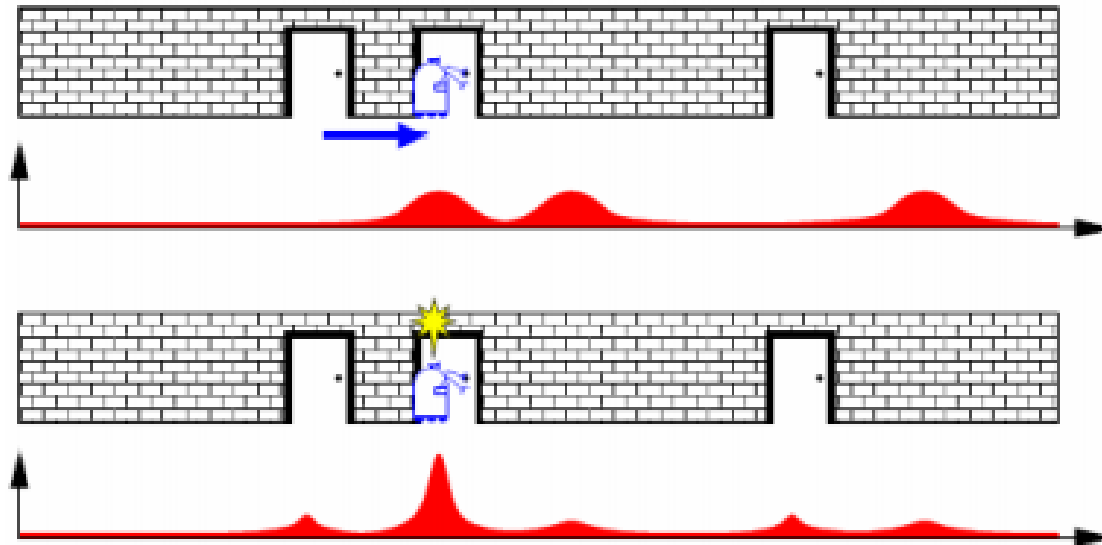


https://www.cs.cmu.edu/~motionplanning/lecture/Chap9-Bayesian-Mapping_howie.pdf

Localization

➤ Localization & Mapping

- 1D-localization example
 - ✓ Detect the land mark
 - ✓ Estimate the distribution of positions

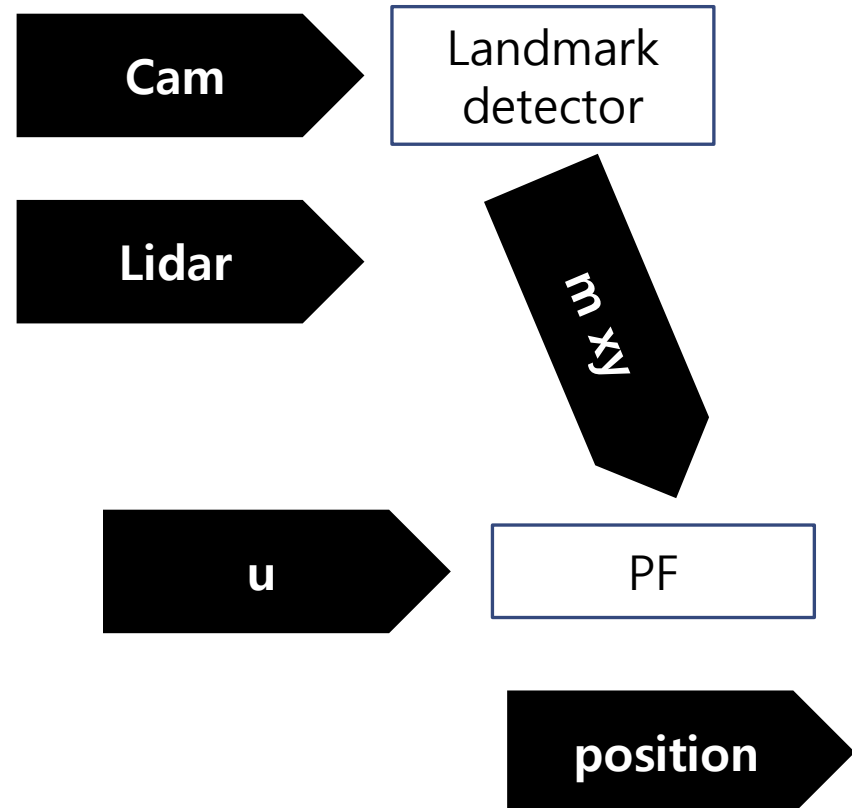


https://www.cs.cmu.edu/~motionplanning/lecture/Chap9-Bayesian-Mapping_howie.pdf

Localization

➤ Localization with PF

- 전체 프로세스



Localization

➤ Localization with PF

- Map 정의

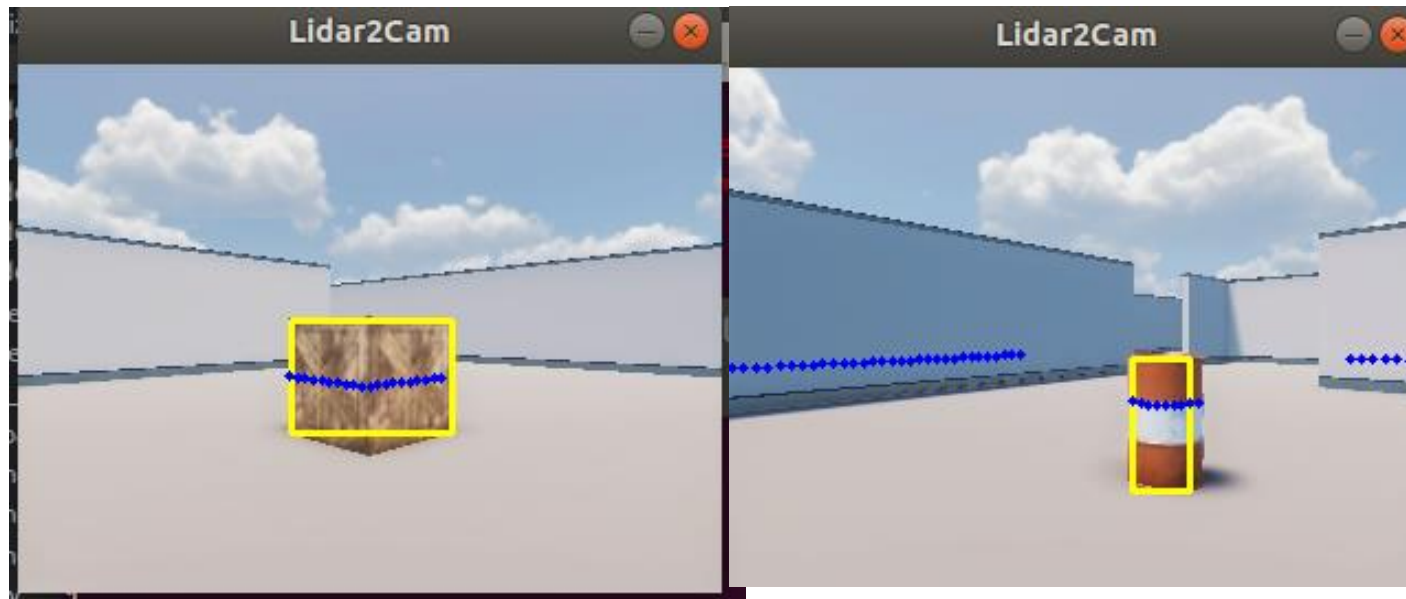
- ✓ 시뮬레이터라서 land mark ground truth 를 알 수 있음
- ✓ Object info 대로 land mark 의 좌표 리스트를 맵으로 사용



Localization

➤ Localization with PF

- Landmark detection
 - ✓ 지난 시간에 했던 object detection 및 extrinsic calibration 코드 사용
 - ✓ 서로 다른 landmark 끼리 분류가 되어 한다



Localization

➤ Localization with PF

- Landmark detection
 - ✓ Landmark는 이 두 배럴을 사용하고자 한다.



Localization

➤ Localization with PF

- Landmark detection node
 - ✓ extrinsic calibration 코드 안에 다음과 같이 object detection 을 넣는다.
 - ✓ Landmark 좌표를 publish 하는 lm_pub 정의

```
class OBDetect:
    def __init__(self):
        self.image_sub = rospy.Subscriber("/image_jpeg/compressed", CompressedImage, self.callback)
        self.lm_pub = rospy.Publisher("/landmark_local", Float64MultiArray, queue_size=3)
        self.lm_msg = Float64MultiArray()

    def callback(self, msg):
        try:
            np_arr = np.fromstring(msg.data, np.uint8)
            img_bgr = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
        except CvBridgeError as e:
            print(e)

        self.img_hsv = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2HSV)
```

Localization

➤ Localization with PF

- Landmark detection node
 - ✓ Red, yellow barrel을 따로 인지하기 위해 아래와 같이 list 형태로 lower, upper 정의

```
def get_bbox(self):  
    lower_object = [np.array([0,160,90]), np.array([15,160,140])] #red  
    upper_object = [np.array([10,240,220]), np.array([25,240,220])] #yellow  
  
    img_bi_list = []  
    rects = []  
  
    for i in range(len(lower_object)):  
        img_bi = cv2.inRange(self.img_hsv, lower_object[i], upper_object[i])  
        img_bi_list.append(img_bi)
```

Localization

➤ Localization with PF

- Landmark detection node

- ✓ Contour로부터 bbox 를 만들고, box 중심 x component 의 평균 기준에서 많이 떨어져 있는 outlier를 제거
- ✓ Threshold는 80이며, 이 threshold 이내의 bbox들만 추려서 minmax로 x, y, w, h 리스트 저장

```
for i, img_bi in enumerate(img_bi_list):  
    contours, _ = cv2.findContours(img_bi, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)  
    if len(contours)!=0:  
        bbox_tuple = [cv2.boundingRect(cont) for cont in contours]  
        bbox_np = np.array([[x, y, x+w, y+h, x + int(w/2)] for x, y, w, h in bbox_tuple])  
        cx_mean = np.mean(bbox_np[:, 4])  
        bbox_inst_ID = np.arange(bbox_np.shape[0])[abs(bbox_np[:, 4] - cx_mean)<80].tolist()  
        bbox_inst = bbox_np[bbox_inst_ID, :]  
        x1 = np.min(bbox_inst[:, 0])  
        y1 = np.min(bbox_inst[:, 1])  
        x2 = np.max(bbox_inst[:, 2])  
        y2 = np.max(bbox_inst[:, 3])  
        rects.append([x1, y1, x2-x1, y2-y1, i])  
  
return rects
```

Localization

➤ Localization with PF

- Landmark detection node
 - ✓ [landmark 개수, landmark 좌표] 형식의 landmark 리스트를 일렬로 펴서 publish

```
def pub_landmark(self, lm_list):  
    self.lm_msg.data = lm_list.reshape([-1]).tolist()  
    self.lm_pub.publish(self.lm_msg)
```

Localization

➤ Localization with PF

- Landmark detection node

- ✓ Main loop

```
if __name__ == '__main__':  
    rospy.init_node('ex_calib_estimate_xy', anonymous=True)  
  
    ex_calib_transform = SensorCalib()  
  
    ob_detector = OBDetect()  
  
    time.sleep(1)  
  
    rate = rospy.Rate(10)  
  
    while not rospy.is_shutdown():  
        xyz_p = ex_calib_transform.xyz[np.where(ex_calib_transform.xyz[:, 0] >= 0)]  
        xyz_c = ex_calib_transform.l2c_trans.transform_lidar2cam(xyz_p)  
        xy_i = ex_calib_transform.l2c_trans.project_pts2img(xyz_c, False)  
        xyii = np.concatenate([xy_i, xyz_p], axis=1)  
        xyii = ex_calib_transform.l2c_trans.crop_pts(xyii)  
        rects = ob_detector.get_bbox()
```


Localization

➤ Localization with PF

- Landmark detection node
 - ✓ Main loop
 - 예외처리 추가

박스는 잡혔으나, 그 안에 laser scan 이 없을수도 있으며, 이때 nan이 나온다. Nan이 나오면 landmark로 송신하지 않도록 한다.

```
obs_list = []

if len(rects)!=0:
    for (x, y, w, h, id_o) in rects:
        cx = int(x + w/2)
        cy = int(y + h/2)

        xy_o = xyii[np.logical_and(xyii[:, 0]>=cx-0.5*w, xyii[:, 0]<cx+0.5*w), :]
        xy_o = xy_o[np.logical_and(xy_o[:, 1]>=cy-0.5*h, xy_o[:, 1]<cy+0.5*h), :]

        xy_o = np.mean(xy_o[:, 2:], axis=0)[:2]

        if not np.isnan(xy_o[0]) and not np.isnan(xy_o[1]):
            print(xy_o)

            obs_list.append(xy_o.tolist()+[id_o])

cv2.rectangle(ex_calib_transform.img, (x,y),(x+w,y+h),(0,255,255), 2)
```

Localization

➤ Localization with PF

- Landmark detection node
 - ✓ Main loop

```
obs_list.append(xy_o.tolist()+[id_o])

cv2.rectangle(ex_calib_transform.img, (x,y),(x+w,y+h),(0,255,255), 2)

ob_detector.pub_landmark(np.array(obs_list))

img_l2c = draw_pts_img(ex_calib_transform.img, xy_i[:, 0].astype(np.int32),
                        xy_i[:, 1].astype(np.int32))

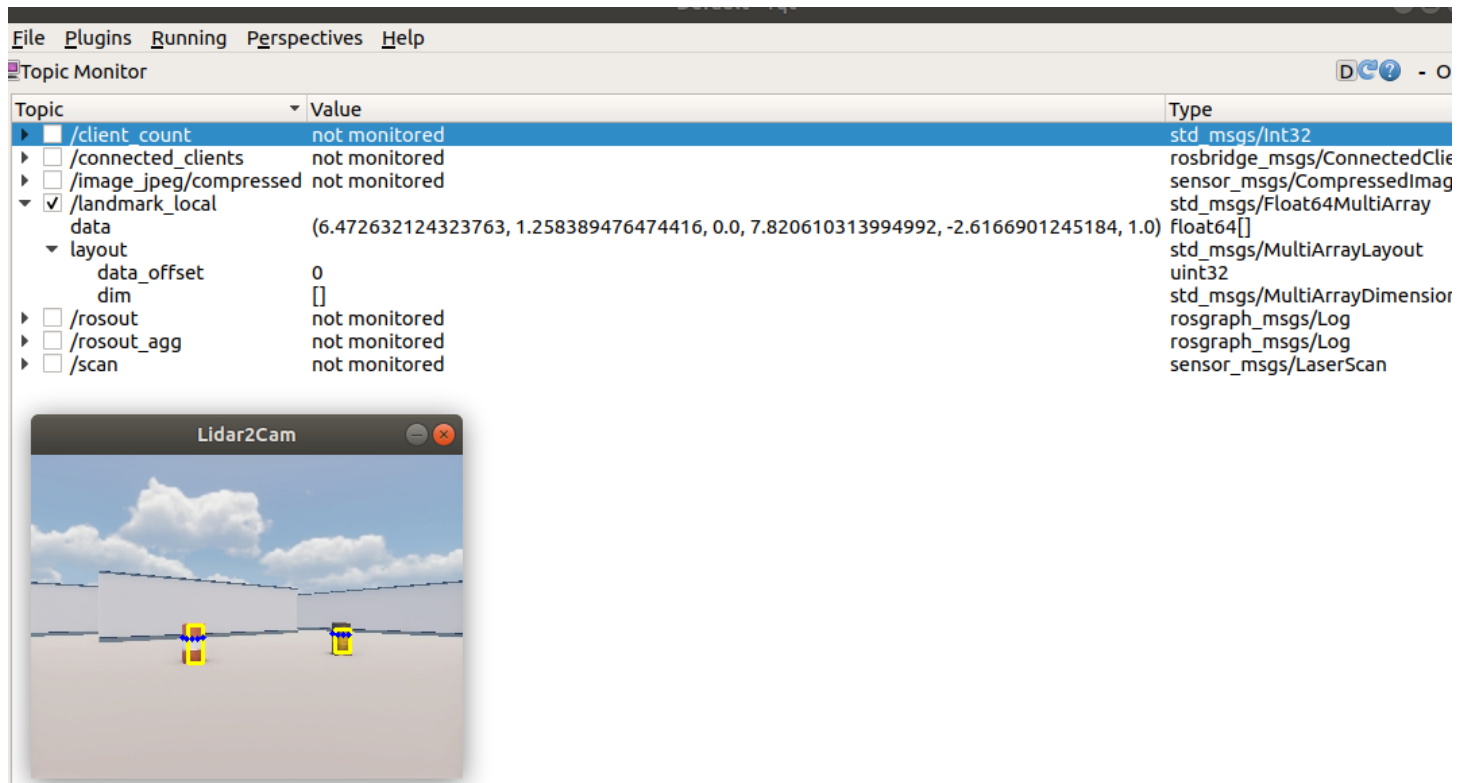
cv2.imshow("Lidar2Cam", img_l2c)
cv2.waitKey(1)
```

Localization

➤ Localization with PF

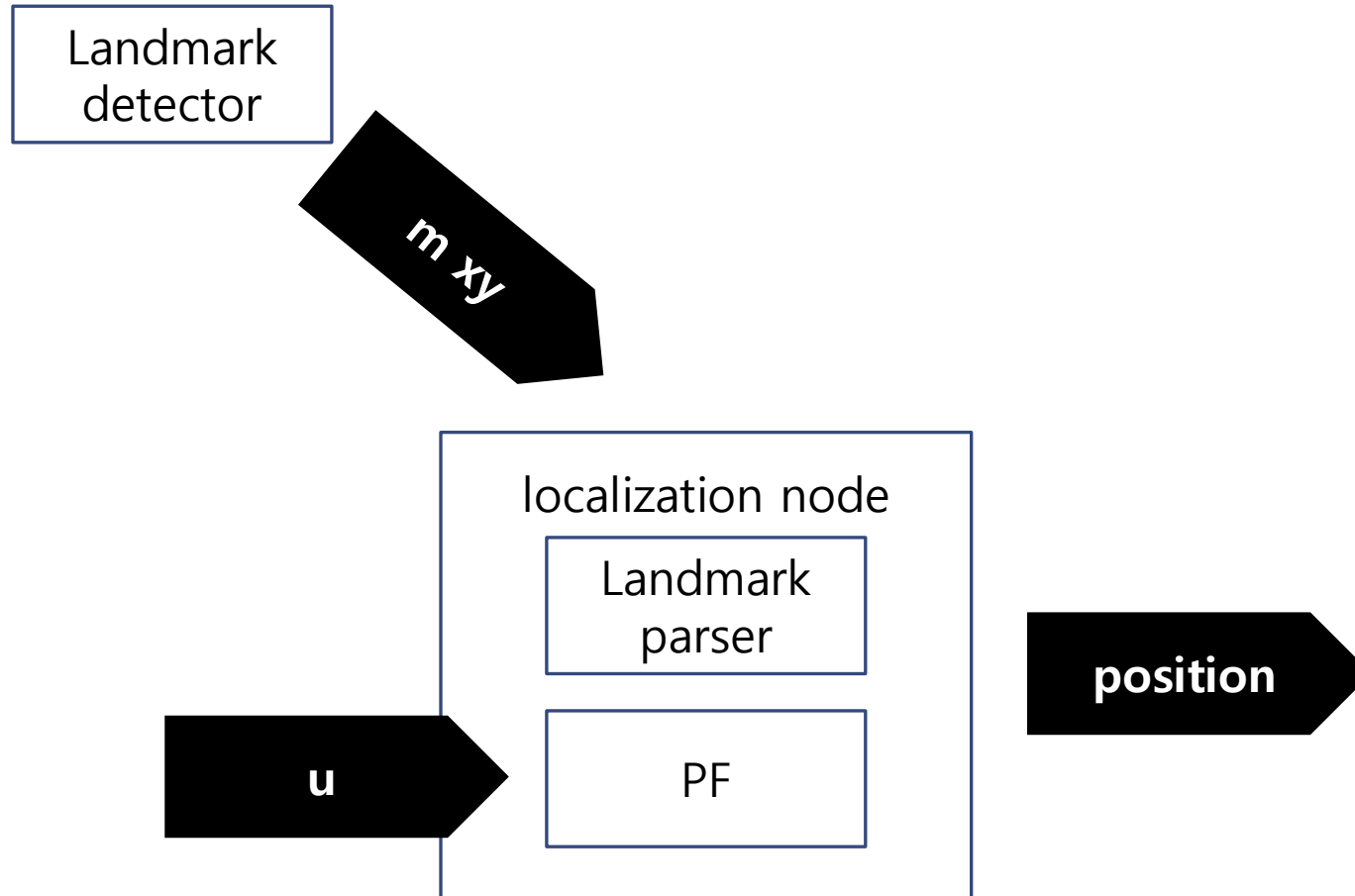
- Landmark detection node

- ✓ 실행 결과



Localization

- Localization with PF
 - Particle filter node



Localization

- Localization with PF
 - Landmark parser

```
class LMparser:
    def __init__(self):
        self.lm_sub = rospy.Subscriber("/landmark_local", Float64MultiArray, self.lm_callback)
        self.lm_measure = None

    def lm_callback(self, msg):
        lm_list = msg.data
        self.lm_measure = lm_list
```

Localization

➤ Localization with PF

- Particle filter class
 - ✓ Land mark list를 맵으로 정의.

```
class ParticleFilter:
    def __init__(self, dt=0.05, NP=3000):

        self.T = dt

        self.NP = NP

        self.XP = np.random.randn(3, self.NP)*20

        self.limit_yaw()

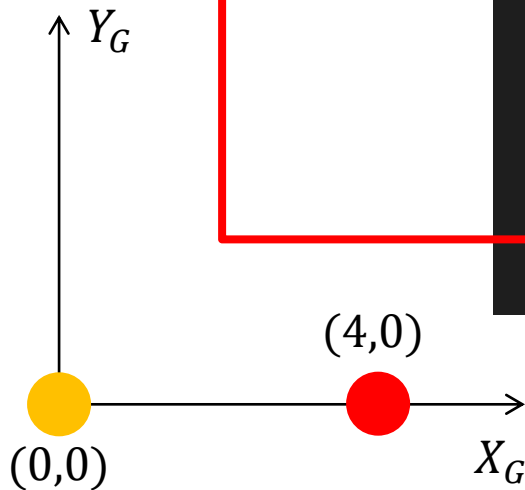
        self.pw = np.ones((NP, ))/NP

        self.q = 0.02

        self.R = np.diag([0.05, 0.05])/self.T

        self.lm_list = [[4.0, 0.0], [0.0, 0.0]]

        self.pose_pub = rospy.Publisher('/pose_pf', PoseArray, queue_size=1)
```



Localization

- Localization with PF
 - Particle filter class

```
def prediction(self, u):  
    dX_pre = np.zeros((3, self.NP), dtype=float)  
    dX_pre[0, :] = u[0]*np.cos(self.XP[2, :])  
    dX_pre[1, :] = u[0]*np.sin(self.XP[2, :])  
    dX_pre[2, :] = u[1] + self.q*np.random.randn(1, self.NP)  
  
    self.XP+=(self.T*dX_pre)  
  
    self.limit_yaw()
```

Localization

➤ Localization with PF

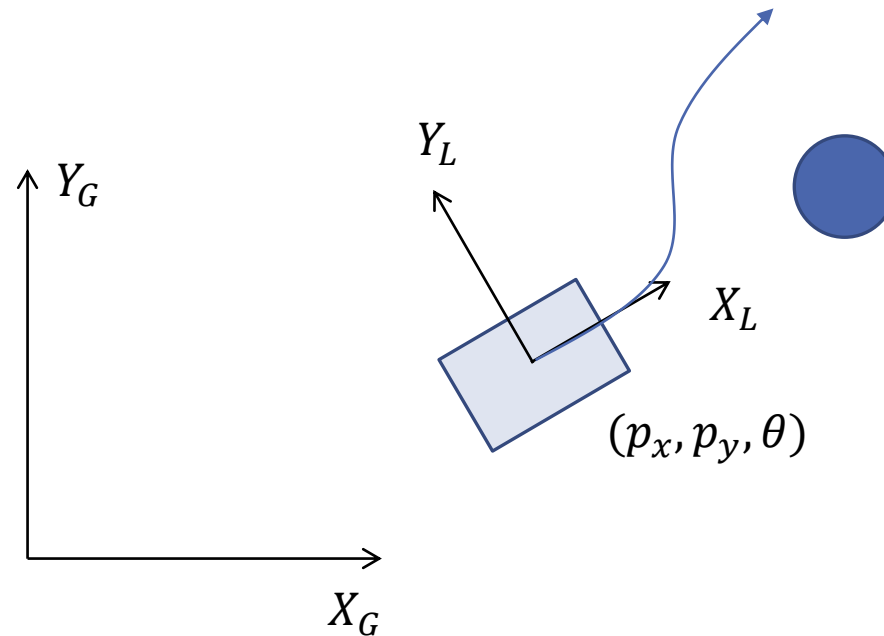
- Particle filter class

```
def limit_yaw(self):  
    over_bool = self.XP[2,:] < -np.pi  
    self.XP[2, over_bool] = self.XP[2, over_bool] + 2*np.pi  
    under_bool = self.XP[2,:] > np.pi  
    self.XP[2, under_bool] = self.XP[2, under_bool] - 2*np.pi  
  
def re_sampling(self, w):  
    re_sample_ID = np.random.choice(np.arange(self.NP), self.NP, p=w)  
    w = np.ones((self.NP, ))/self.NP  
    return re_sample_ID, w
```


Localization

➤ Localization with PF

- Landmark measurement model
 - ✓ 2d vehicle 좌표 변환

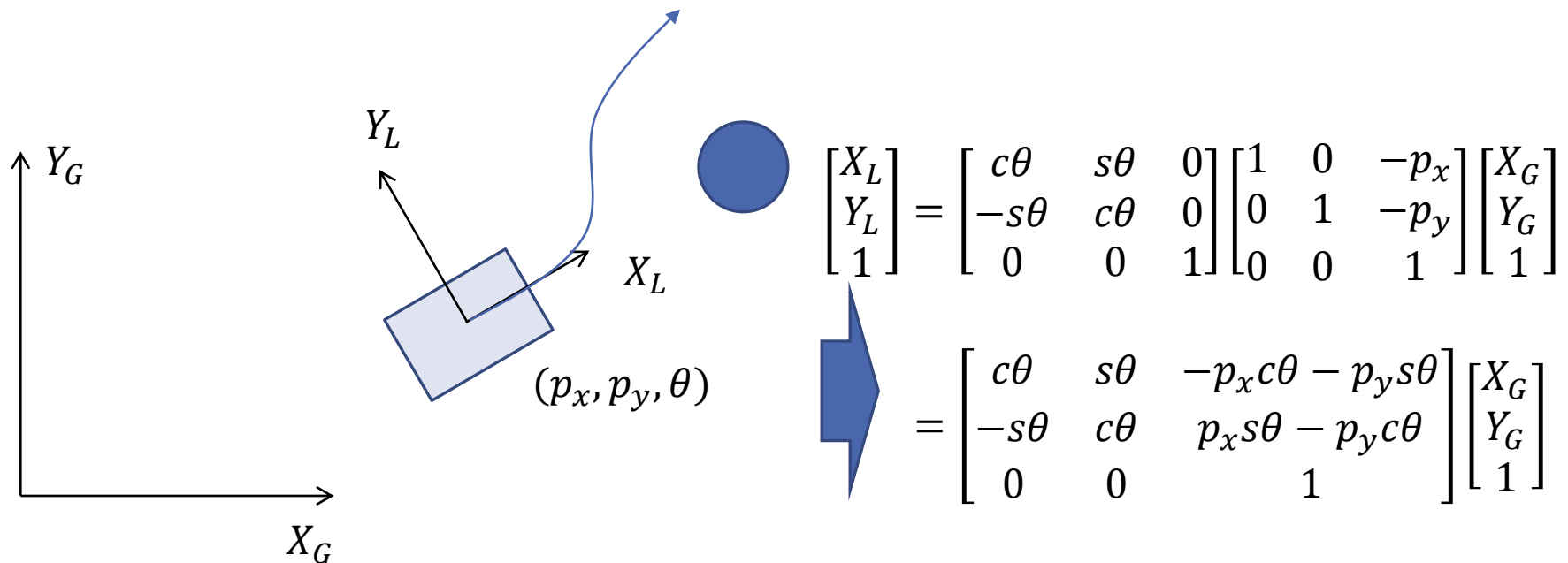


$$\begin{bmatrix} X_1 \\ Y_1 \\ 1 \end{bmatrix}_G = \begin{bmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_2 \\ Y_2 \\ 1 \end{bmatrix}_L$$

Localization

➤ Localization with PF

- Landmark measurement model
 - ✓ 2d vehicle 좌표 변환



$$\begin{bmatrix} X_G \\ Y_G \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_L \\ Y_L \\ 1 \end{bmatrix}$$

➤ Localization with PF

- Landmark measurement model
 - ✓ 차량 기준 landmark measurement model

$$\begin{aligned} \begin{bmatrix} L_x \\ L_y \end{bmatrix} &= \begin{bmatrix} c\theta & s\theta & -p_x c\theta - p_y s\theta \\ -s\theta & c\theta & p_x s\theta - p_y c\theta \end{bmatrix} \begin{bmatrix} X_M \\ Y_M \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} X_M c\theta + Y_M s\theta - p_x c\theta - p_y s\theta \\ -X_M s\theta + Y_M c\theta + p_x s\theta - p_y c\theta \end{bmatrix} \end{aligned}$$

Localization

➤ Localization with PF

- Landmark measurement model
 - ✓ landmark measurement model method

```
def landmark_measurement_model(self, lm_g):  
  
    cth = np.cos(self.XP[2, :])  
    sth = np.sin(self.XP[2, :])  
  
    lm_local = np.zeros((2, self.NP))  
  
    lm_local[0, :] = lm_g[0]*cth + lm_g[1]*sth - self.XP[0, :]*cth - self.XP[1, :]*sth  
    lm_local[1, :] = -lm_g[0]*sth + lm_g[1]*cth + self.XP[0, :]*sth - self.XP[1, :]*cth  
  
    return lm_local
```

Localization

➤ Localization with PF

- Correction with landmarks

Landmark array를 받아서
reshape. [x, y, id]

Lm에서 [id]만
골라온다.
ex) 노란색이면
id==1

파티클 기준에서
관측된 lm의 스카우트
미니 기준 x,y를 구함

관측값에 가까운
파티클에 weight 높게
준다.

각 lm 관측에 대해서는
and 관계이므로 확률은
곱해준다.

```
def correction(self, Z):  
    if len(Z)!=0:  
        Z = np.array(Z).reshape([-1, 3])  
        pw = self.pw  
        for i in range(Z.shape[0]):  
            id_lm = int(Z[i, 2])  
            lm_local = self.landmark_measurement_model(self.lm_list[id_lm])  
            dz = lm_local-Z[i, :2].reshape([2, -1])  
            pdf_z = pdf_multivariate_gauss(dz, self.R)  
            pw = pw*pdf_z  
        pw = pw / pw.sum()
```

Localization

➤ Localization with PF

- Correction with landmarks
 - ✓ 그 외의 건 이전과 동일

Land mark 관측
안됐으면 pass

```
self.Xm = np.dot(self.XP, pw).reshape([-1, 1])

Xcov = np.dot([
    (self.XP-self.Xm),
    np.diag(pw)
]).dot((self.XP-self.Xm).T)

re_sample_ID, self.pw = self.re_sampling(pw)

XP_new = self.XP[:, re_sample_ID]

self.XP = XP_new + np.diag([0.2, 0.2, 0.01]).dot(np.random.randn(3, self.NP))

print(self.Xm)

self.limit_yaw()

else:

    pass
```

Localization

➤ Localization with PF

- Main loop

```
✓ if __name__ == '__main__':  
    rospy.init_node('PF_localization', anonymous=True)  
  
    rate = rospy.Rate(20)  
  
    pf = ParticleFilter(dt=0.05)  
  
    cmd_gen = CMDPublisher()  
  
    lm_parser = LMparser()
```

Localization

➤ Localization with PF

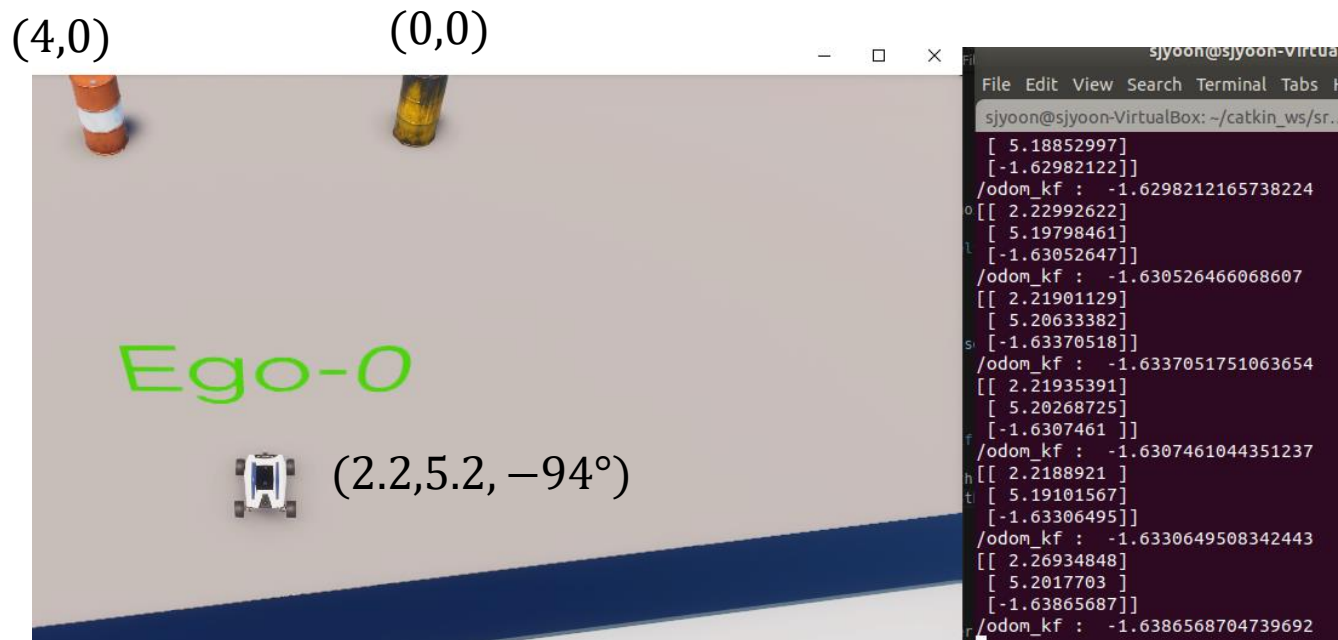
- Main loop

```
while not rospy.is_shutdown():  
    if lm_parser.lm_measure is not None:  
        #decide the u  
        u = cmd_gen.calc_u()  
  
        #prediction step  
        pf.prediction(u)  
  
        #measurement locations  
        z = lm_parser.lm_measure  
  
        #correction step  
        pf.correction(z)  
  
        #get the estimated states  
        pf.send_estimated_state()  
    else:  
        pass  
  
    rate.sleep()
```


Localization

➤ Localization with PF

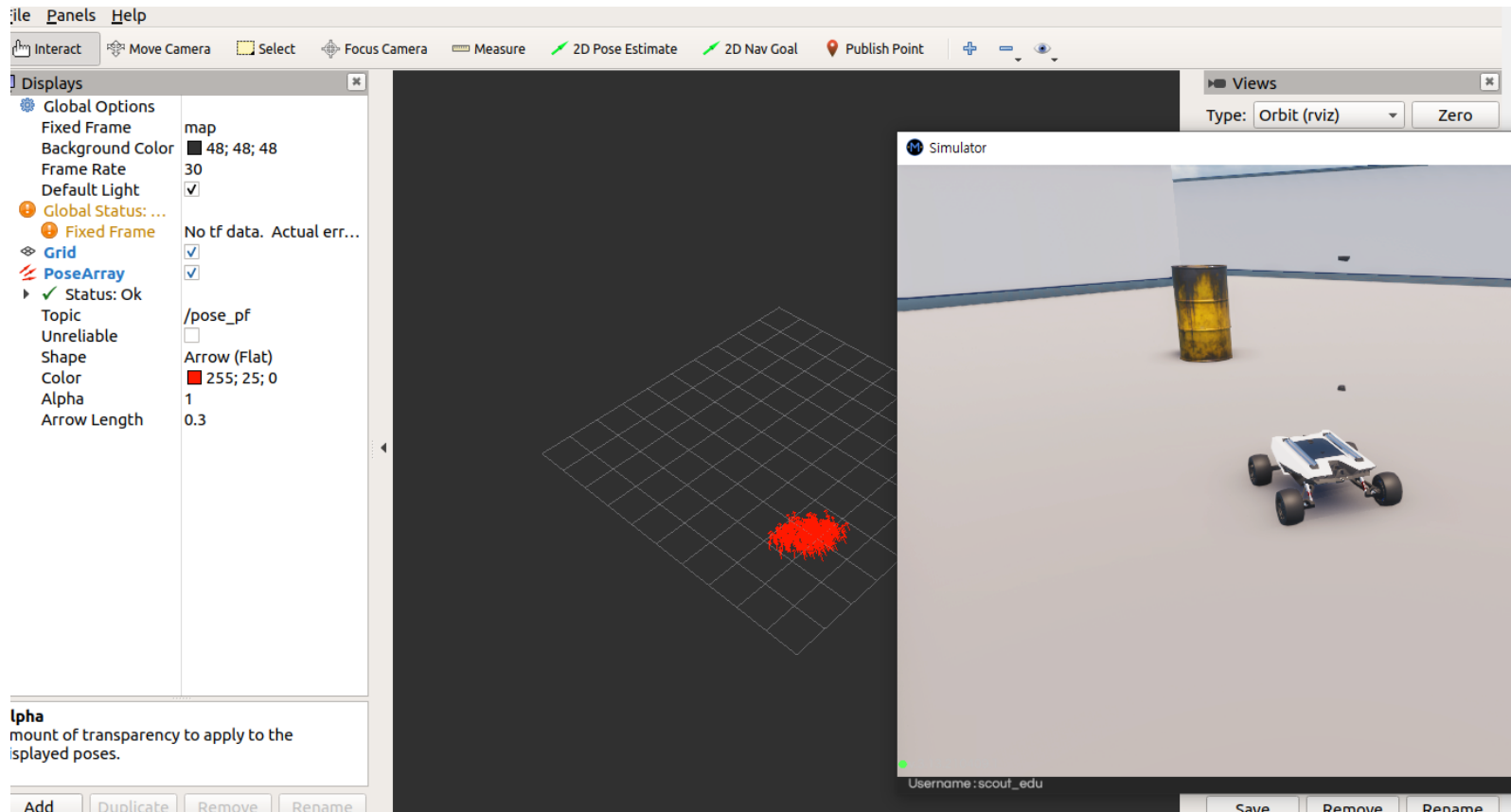
- Localization
 - ✓ 실행 결과



Localization

➤ Localization with PF

- Localization
 - ✓ 실행 결과



Localization

➤ Localization with PF

- Localization
 - ✓ 실행 결과

