

# SC649 Project

“EKF localization to move around in an uncertain environment”



By

▶ Shailesh K. Mahindrakar (24D1178)

▶ Krishna N Agaram (210051003)

▶ Govind Kumar (210050058)

▶ Under the supervision of

▶ Prof. Leena Vachchani

# Q1. Calculation of robot pose using EKF

## Extended Kalman Filter Algorithm

- Initialize with  $x_0$
- At each time step  $n$

1 Predict

$$\hat{x}_{n,\text{prior}} = f(\hat{x}_{n-1,\text{update}})$$
$$P_{n,\text{prior}} = F P_{n-1,\text{update}} F^T + Q$$

2 Update

$$K_n = P_{n,\text{prior}} H^T (H P_{n,\text{prior}} H^T + R_{\text{measure}})^{-1}$$
$$\hat{x}_{n,\text{update}} = \hat{x}_{n,\text{prior}} + K_n [z_{n,\text{measure}} - h(\hat{x}_{n,\text{prior}})]$$
$$P_{n,\text{update}} = (I - K_n H) P_{n,\text{prior}}$$

# Implementation Details (1/4): Prediction

- ▶ `predicted_pose = predict_state( estimated_pose );`
- ▶  $P = F @ P @ F.T + Q$
- ▶ We use  $X_t = X_{t-1} + [v \cos(\theta), v \sin(\theta), w] * K_{\text{samp}}$ , where  $[v, w]$  is the control input (standard unicycle situation) and  $K_{\text{samp}}$  the sampling frequency
- ▶ Here, `estimate_pose` represents  $\mu_{t-1}$  and `predicted_pose` represents  $b\mu_t$  (b for bar, i.e. predicted  $\mu_t$ )
- ▶  $F$  is the linearized state dynamics function, roughly identity.  $P$  after assignment is the predicted\_covariance of  $x$  at time  $t$ .

# Implementation Details (2/4): measurement and residual



predicted\_measurement =  
predict\_measurement(our\_predicted\_pose, landmarkA,  
landmarkB, landmarkC)



residual =  $Y_{\text{measured}}$  - predicted\_measurement



Here, predicted\_measurement corresponds to  $h(\mu_{t-1})$  and  $Y_{\text{measured}}$  to  $z_t$ .



So the residual is  $z_t - h(\mu_{t-1})$ .

# Implementation details (3/4): Kalman Gain



$H_t = \text{get\_current\_H}(\text{our\_predicted\_pose}, \text{landmarkA}, \text{landmarkB}, \text{landmarkC})$



$PH_T = \text{np.matmul}(P, H_t.T)$



$S = \text{np.matmul}(H_t, PH_T) + R$



$S_{\text{inv}} = \text{np.linalg.inv}(S)$



$\text{filter\_gain} = \text{np.matmul}(PH_T, S_{\text{inv}})$



$H_t$  represents the derivative of the measurement function  $h(\cdot)$ .  
 $\text{filter\_gain}$  corresponds to the Kalman Gain.

# Implementation Details (4/4): Update step

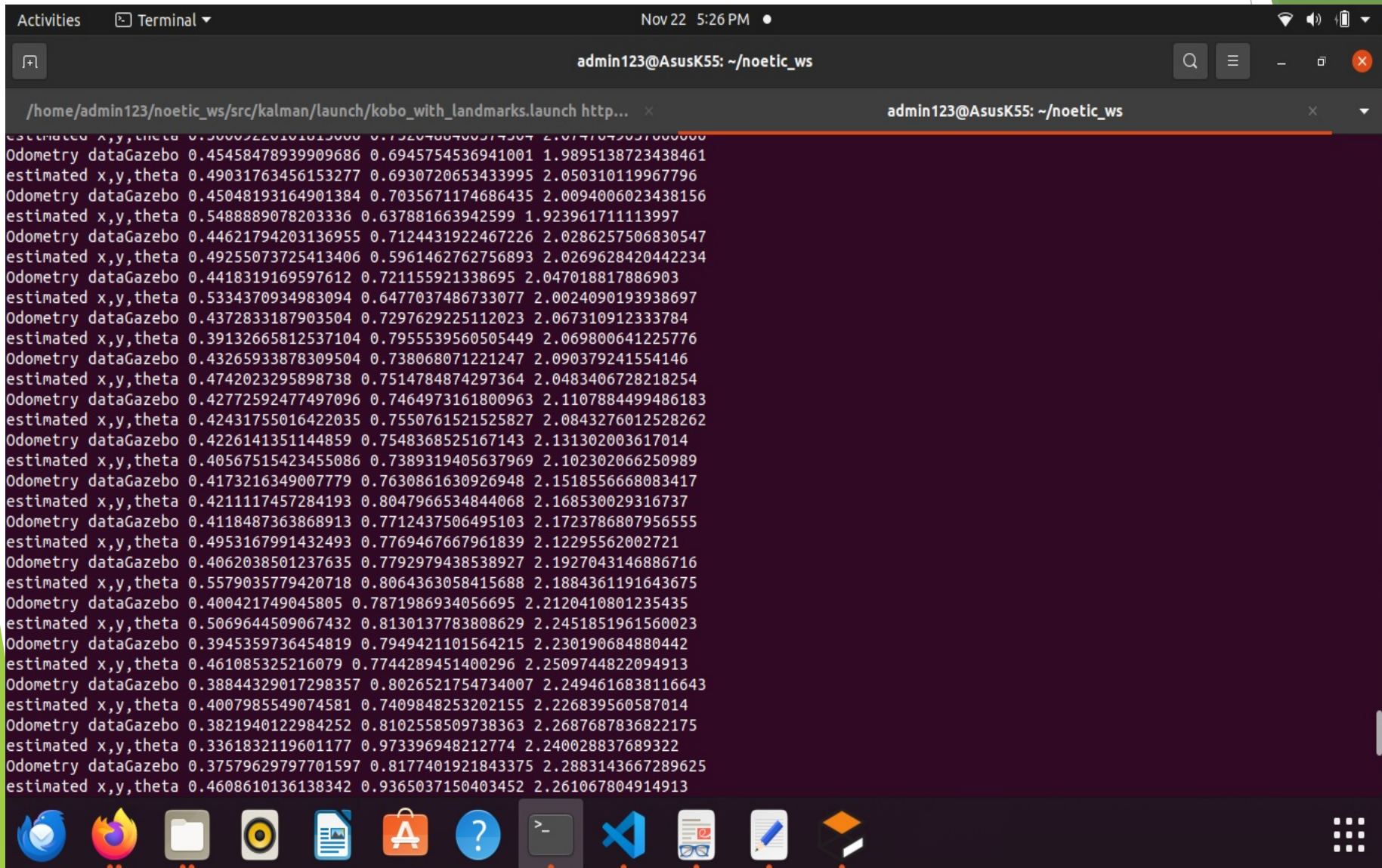
Finally, we update the predictions into estimates for position at time  $t$ .

```
estimated_pose = our_predicted_pose +  
np.matmul(filter_gain, residual)
```

```
P = (I - np.matmul(filter_gain, H_t))@P
```

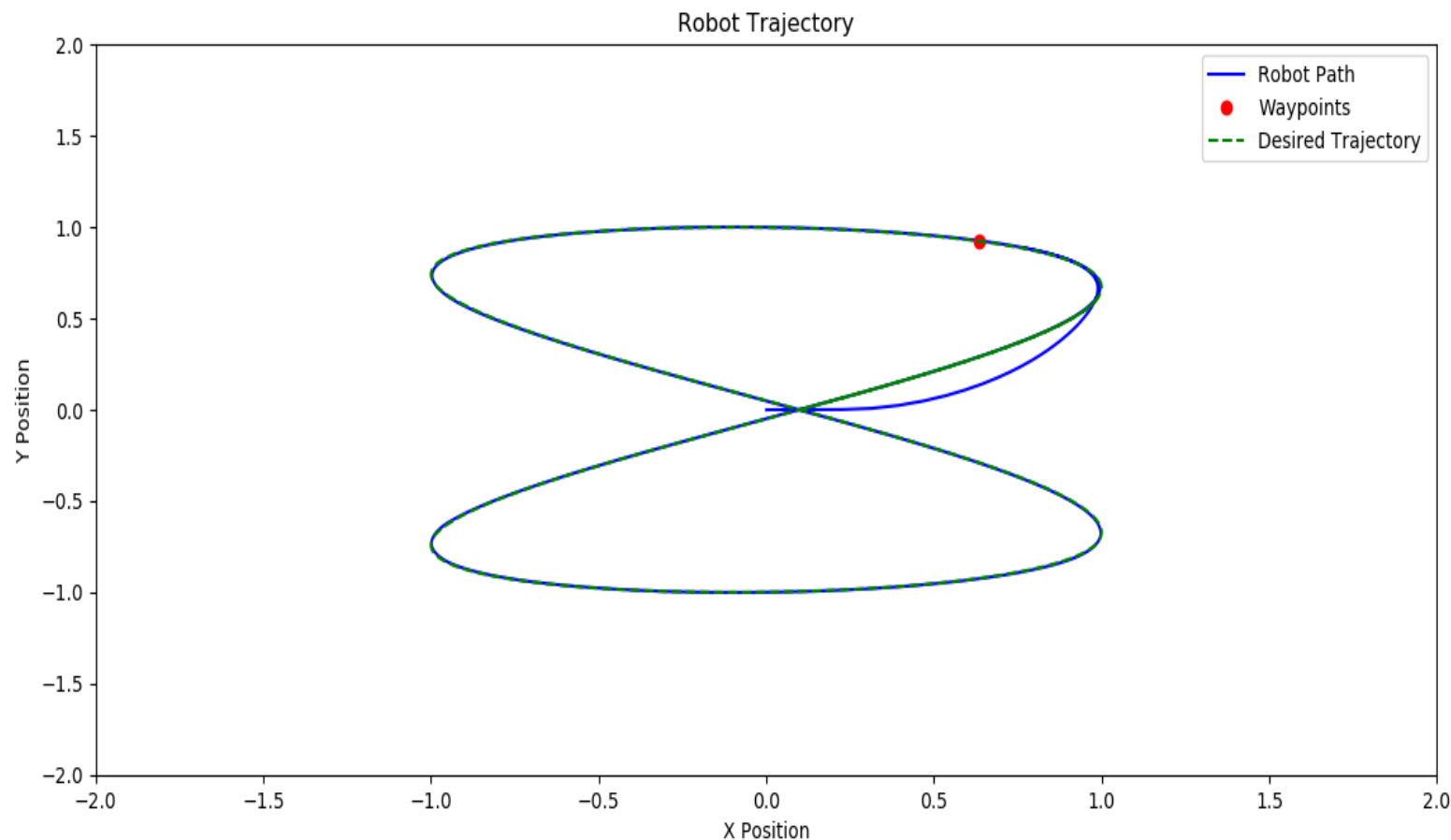
Estimated\_pose =  $\mu_t$ , estimated covariance is  $P$ .

# EKF works – we are able to capture almost the same odometry

A terminal window titled 'admin123@AsusK55: ~/noetic\_ws' displays a list of odometry data. The data is organized into pairs of 'estimated' and 'Odometry dataGazebo' entries. Each entry contains three columns of numerical values representing x, y, and theta coordinates. The terminal window has a dark theme and a standard Ubuntu desktop environment is visible in the background.

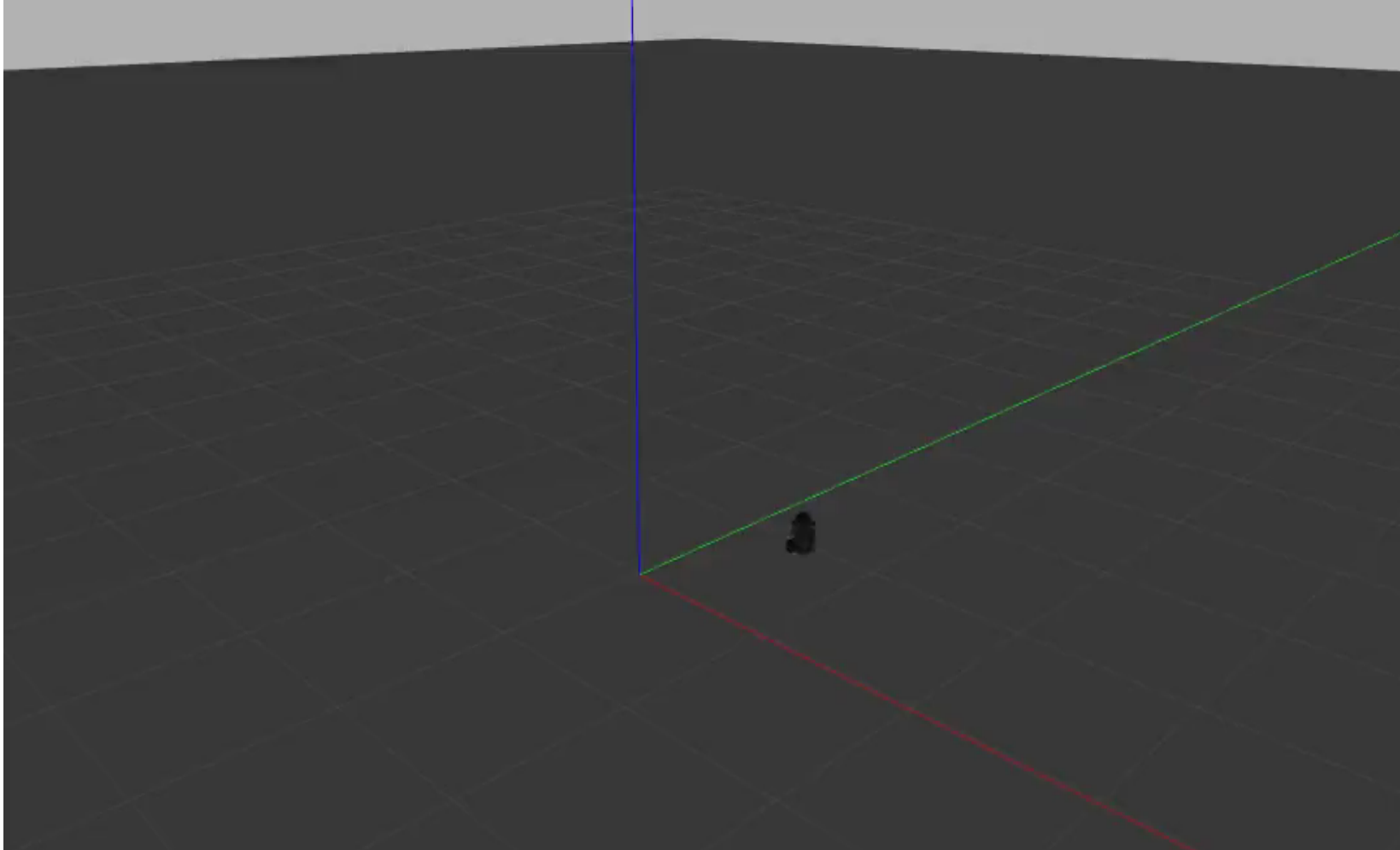
```
admin123@AsusK55: ~/noetic_ws  
/home/admin123/noetic_ws/src/kalman/launch/kobo_with_landmarks.launch http...  
estimated x,y,theta 0.3000322010101000 0.7320400400374304 2.0147042037000000  
Odometry dataGazebo 0.45458478939909686 0.6945754536941001 1.9895138723438461  
estimated x,y,theta 0.49031763456153277 0.6930720653433995 2.050310119967796  
Odometry dataGazebo 0.45048193164901384 0.7035671174686435 2.0094006023438156  
estimated x,y,theta 0.5488889078203336 0.637881663942599 1.923961711113997  
Odometry dataGazebo 0.44621794203136955 0.7124431922467226 2.0286257506830547  
estimated x,y,theta 0.49255073725413406 0.5961462762756893 2.0269628420442234  
Odometry dataGazebo 0.4418319169597612 0.721155921338695 2.047018817886903  
estimated x,y,theta 0.5334370934983094 0.6477037486733077 2.0024090193938697  
Odometry dataGazebo 0.4372833187903504 0.7297629225112023 2.067310912333784  
estimated x,y,theta 0.39132665812537104 0.7955539560505449 2.069800641225776  
Odometry dataGazebo 0.43265933878309504 0.738068071221247 2.090379241554146  
estimated x,y,theta 0.4742023295898738 0.7514784874297364 2.0483406728218254  
Odometry dataGazebo 0.42772592477497096 0.7464973161800963 2.1107884499486183  
estimated x,y,theta 0.42431755016422035 0.7550761521525827 2.0843276012528262  
Odometry dataGazebo 0.4226141351144859 0.7548368525167143 2.131302003617014  
estimated x,y,theta 0.40567515423455086 0.7389319405637969 2.102302066250989  
Odometry dataGazebo 0.4173216349007779 0.7630861630926948 2.1518556668083417  
estimated x,y,theta 0.4211117457284193 0.8047966534844068 2.168530029316737  
Odometry dataGazebo 0.4118487363868913 0.7712437506495103 2.1723786807956555  
estimated x,y,theta 0.4953167991432493 0.7769467667961839 2.12295562002721  
Odometry dataGazebo 0.4062038501237635 0.7792979438538927 2.1927043146886716  
estimated x,y,theta 0.5579035779420718 0.8064363058415688 2.1884361191643675  
Odometry dataGazebo 0.400421749045805 0.7871986934056695 2.2120410801235435  
estimated x,y,theta 0.5069644509067432 0.8130137783808629 2.2451851961560023  
Odometry dataGazebo 0.3945359736454819 0.7949421101564215 2.230190684880442  
estimated x,y,theta 0.461085325216079 0.7744289451400296 2.2509744822094913  
Odometry dataGazebo 0.38844329017298357 0.8026521754734007 2.2494616838116643  
estimated x,y,theta 0.4007985549074581 0.7409848253202155 2.226839560587014  
Odometry dataGazebo 0.3821940122984252 0.8102558509738363 2.2687687836822175  
estimated x,y,theta 0.3361832119601177 0.973396948212774 2.240028837689322  
Odometry dataGazebo 0.37579629797701597 0.8177401921843375 2.2883143667289625  
estimated x,y,theta 0.4608610136138342 0.9365037150403452 2.261067804914913
```

## Q2. Trajectory tracking supported with videos and way-points (plotted). Example:

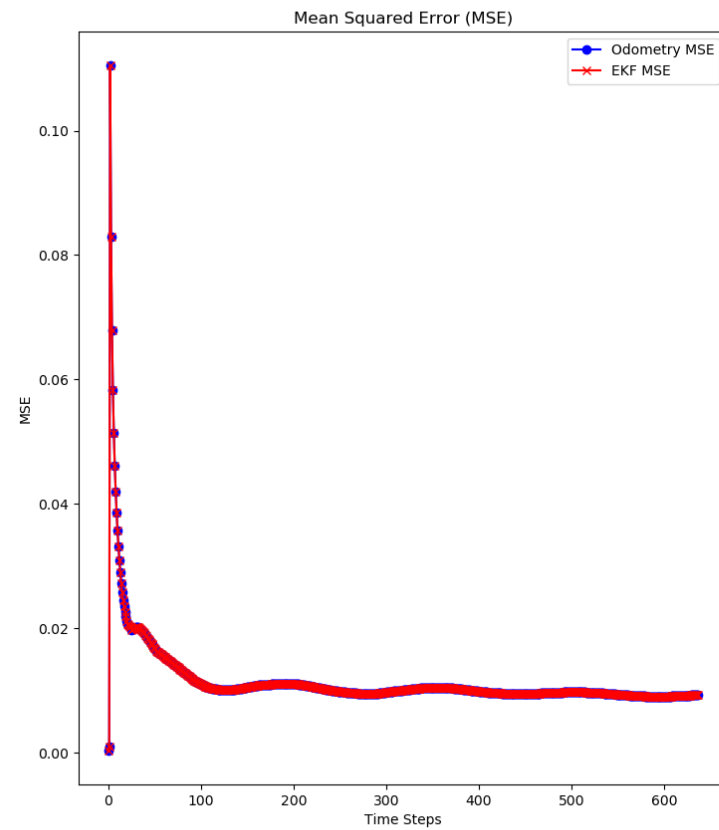
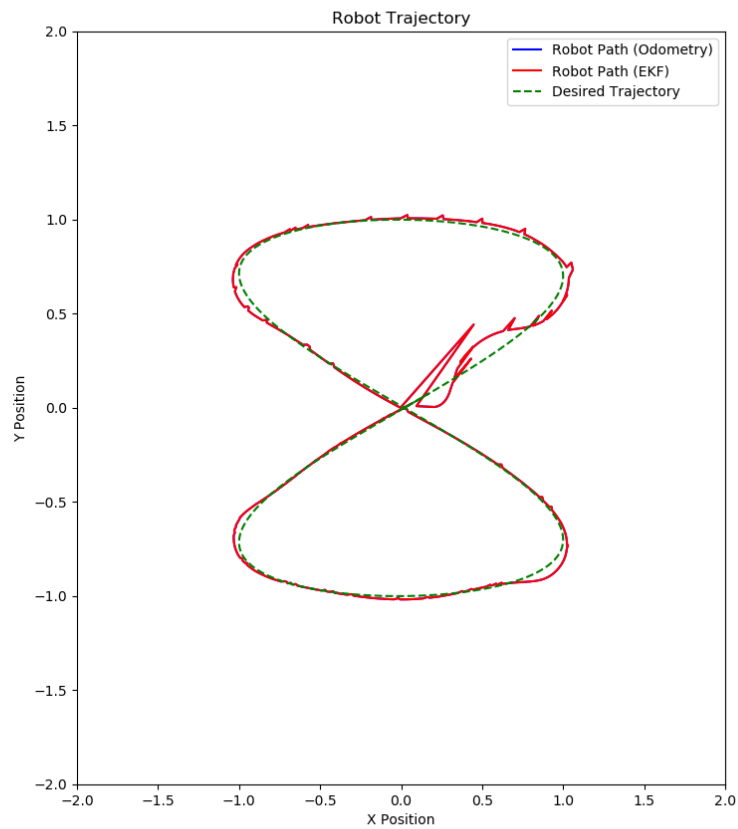




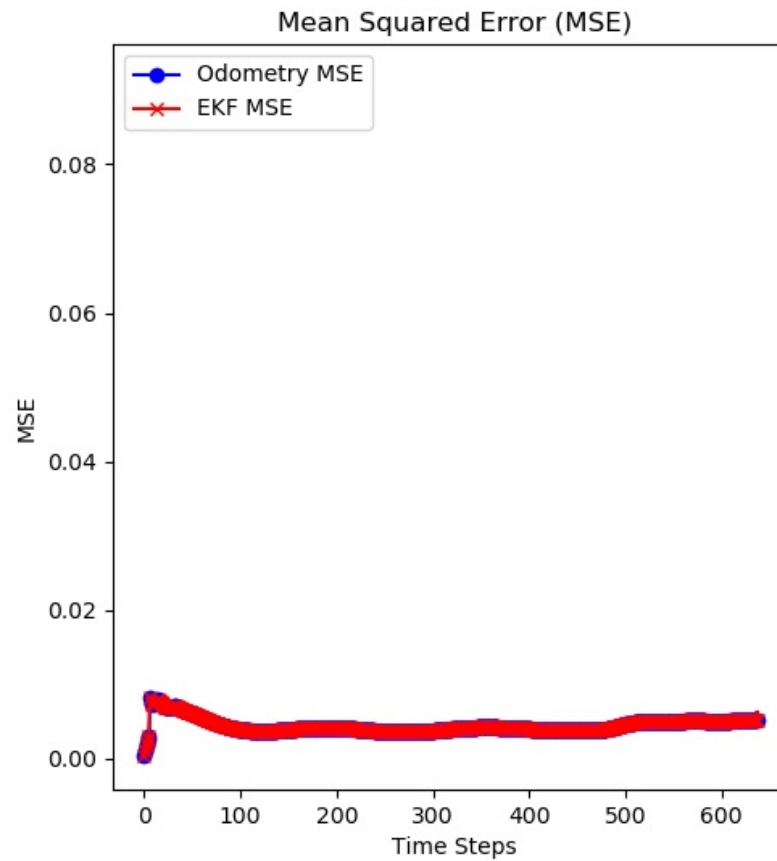
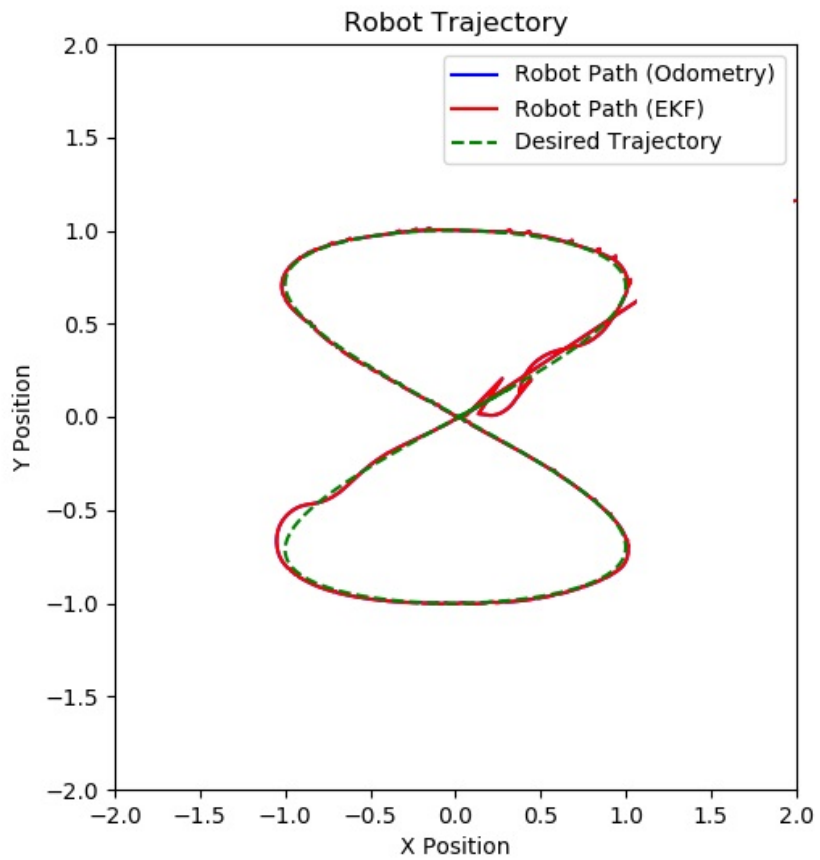
## Corresponding Gazebo video



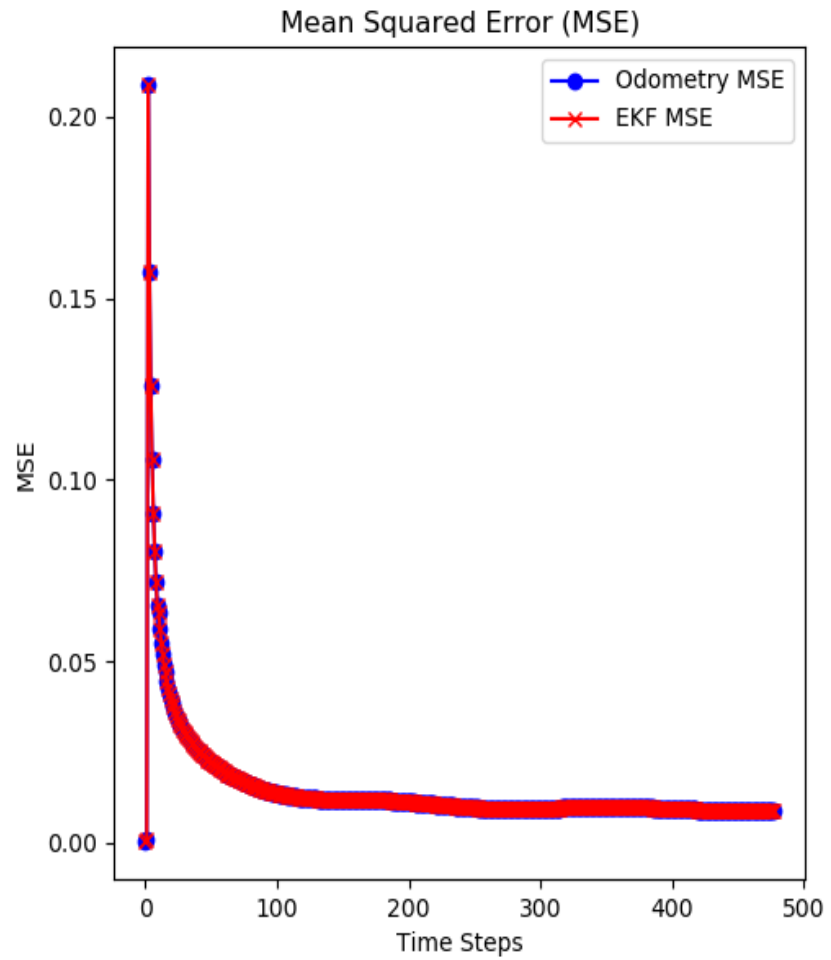
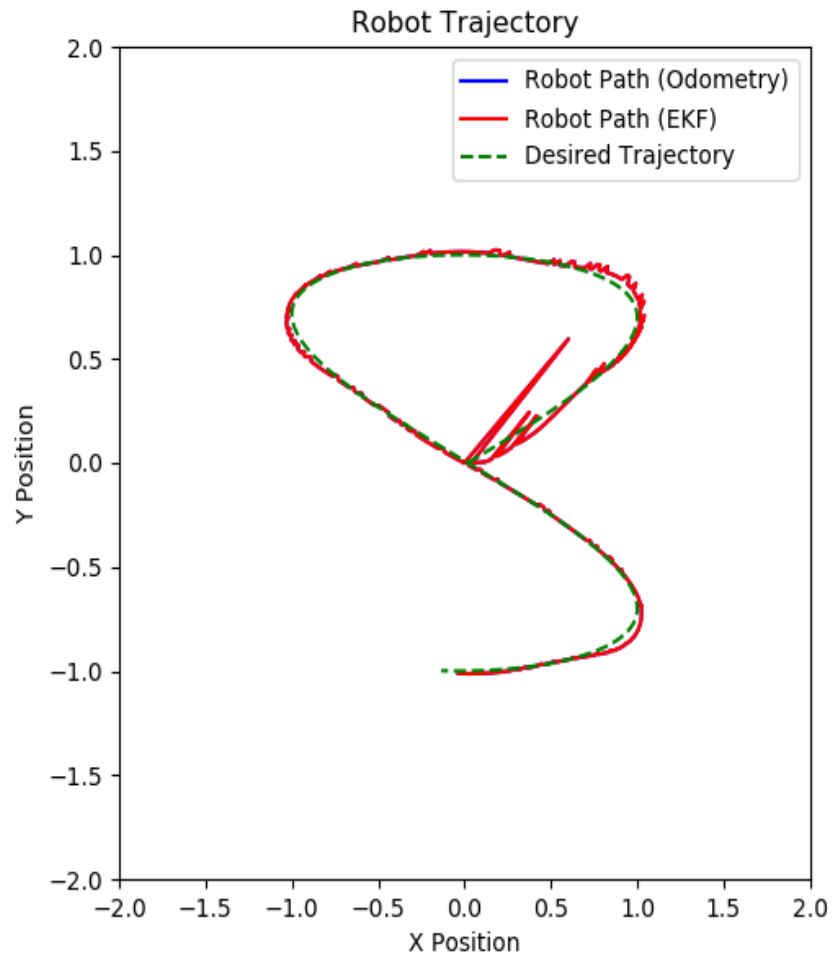
► For  $\delta=0.02 \rightarrow$  MSE @ 0.01



► For  $\delta=0.2 \rightarrow$  MSE @ 0.01



► For  $\delta=2 \rightarrow$  MSE @ 0.03

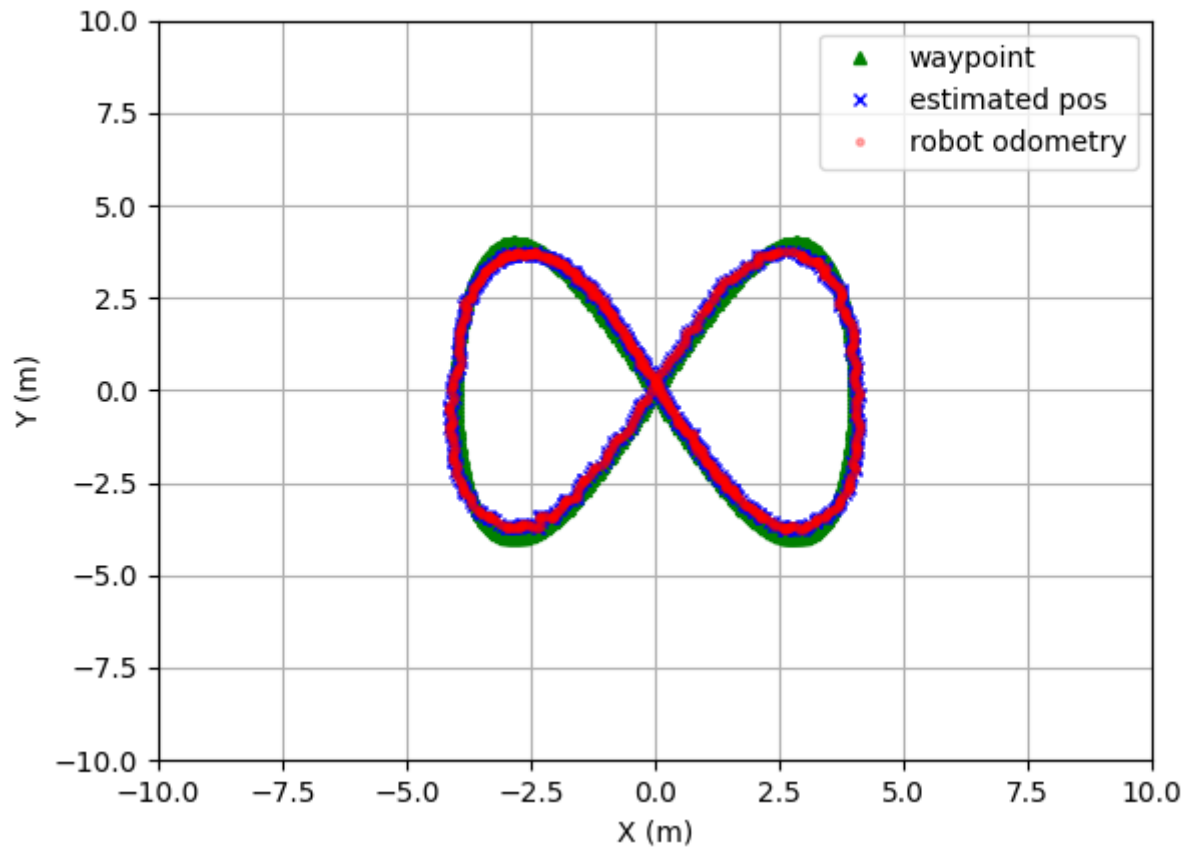


More experiments – we wanted  
to get the Meta logo

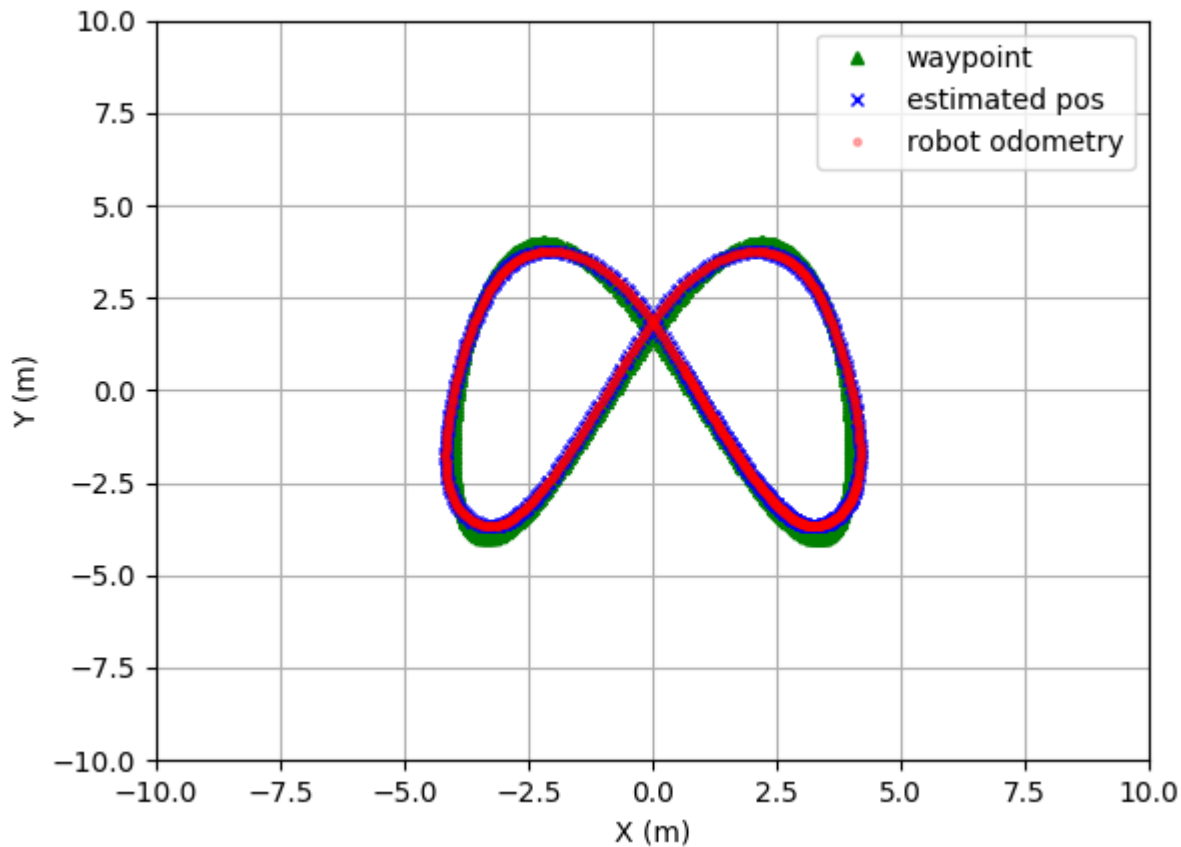
$$X, Y = 4\sin(2t+\delta), 4\cos(3t)$$

We set  $\delta = 0.0, 0.2, 0.4$  and  $0.8$  to see the difference in  
performance (MSE)

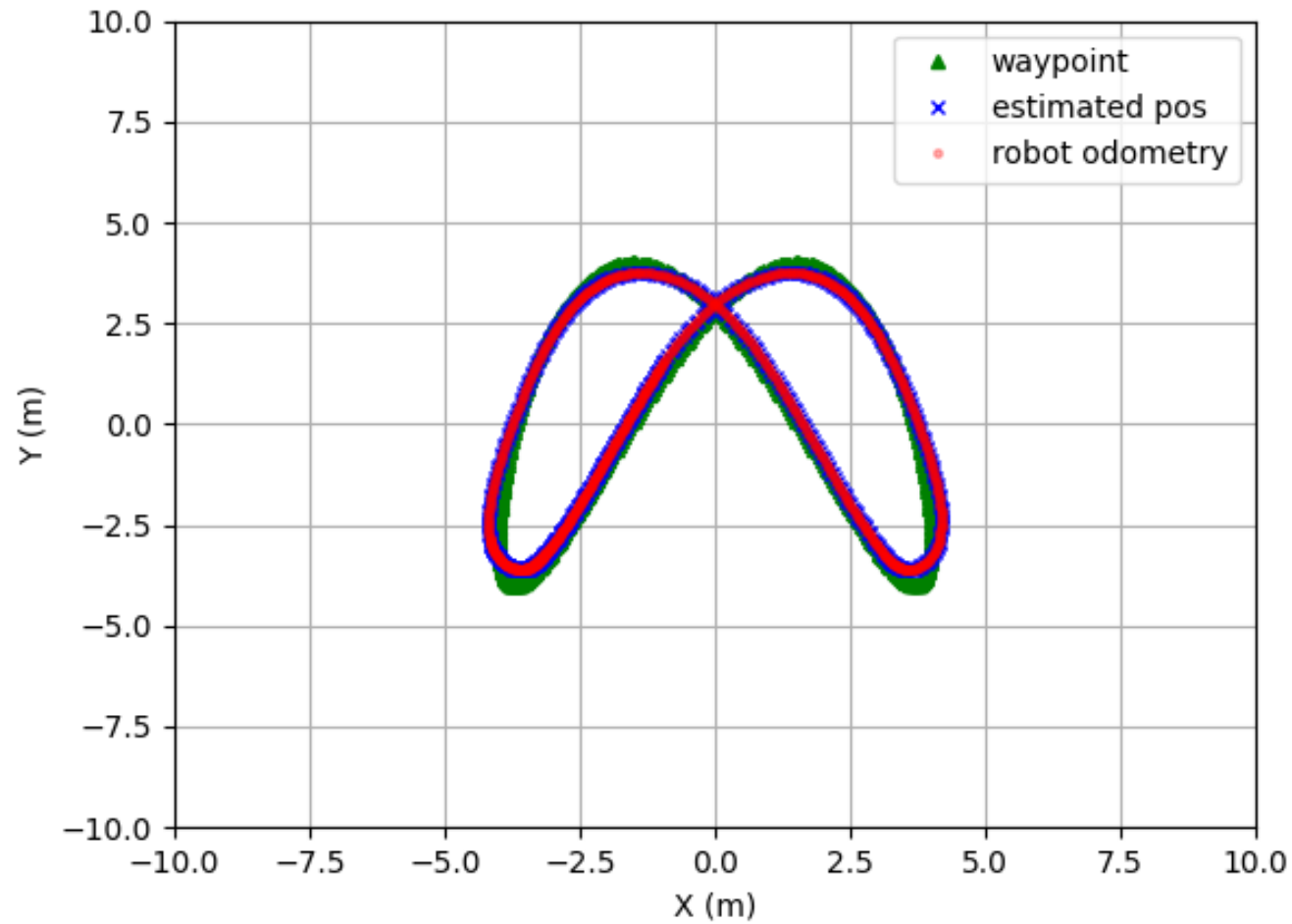
$$\delta = 0; \text{MSE} = .01$$



$$\delta = .2; \text{MSE} = .01$$

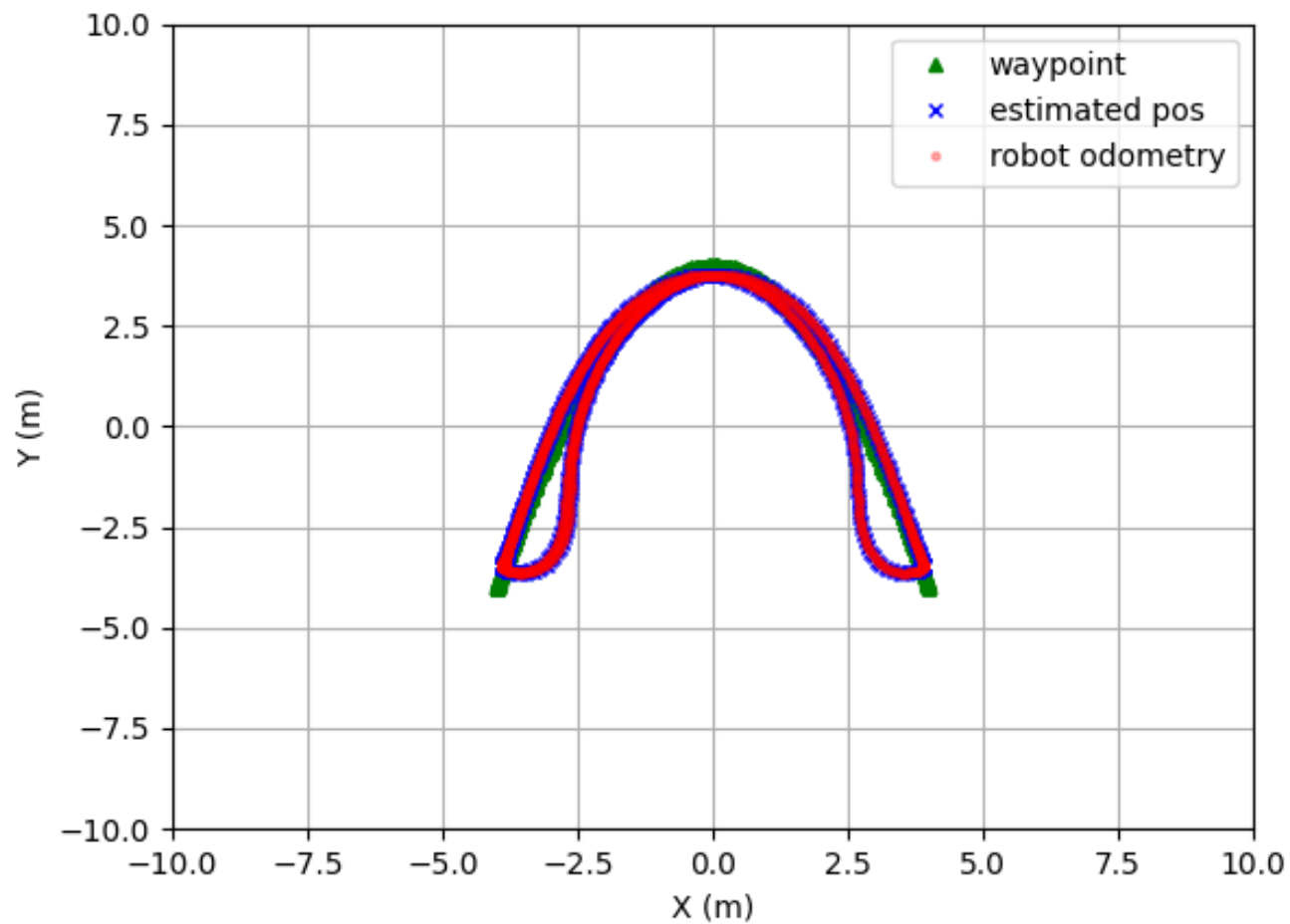


$$\delta = .4; \text{MSE} = .01$$

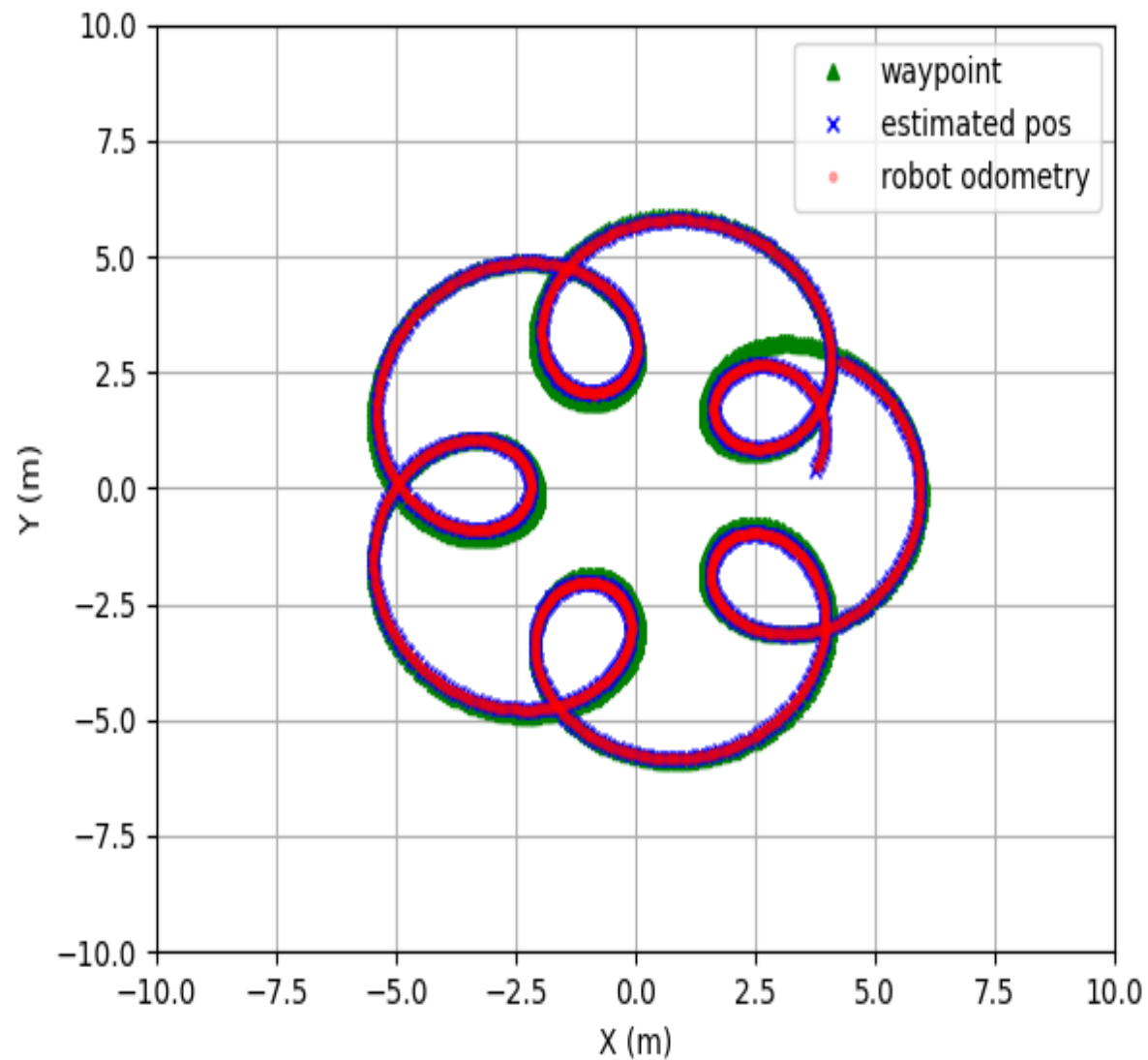


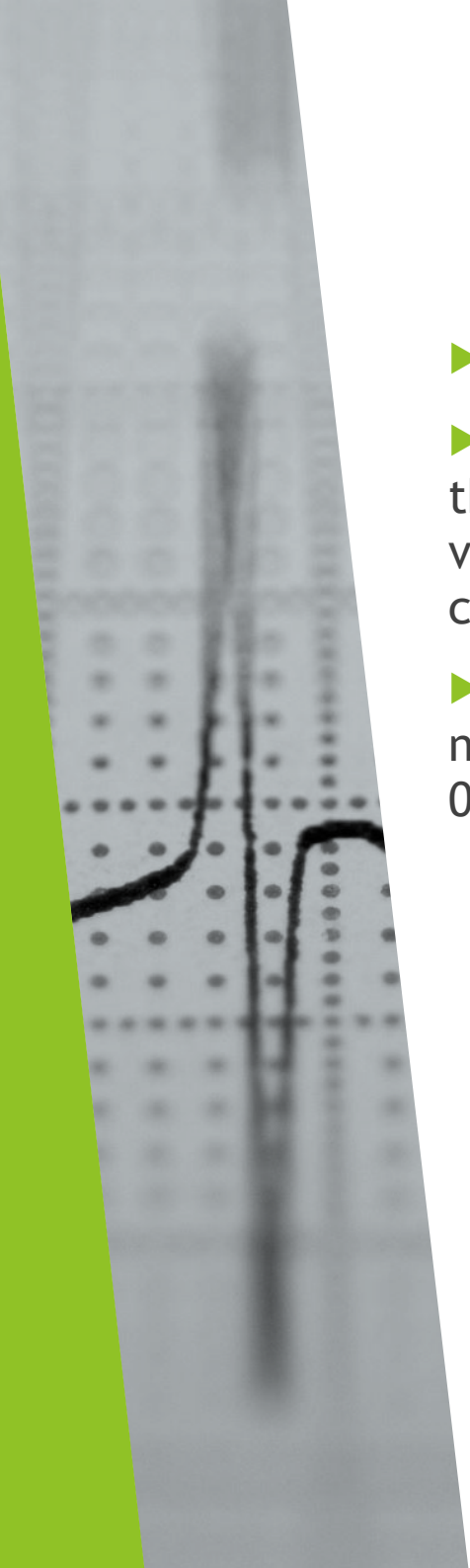


$$\delta = .8; \text{MSE} = .01$$



- All experiments above have their videos on [gdrive](#)
- Another fancy experiment:



- 
- ▶ Notice that the error stabilizes at 0.01 in each case.
  - ▶ This corresponds to the fact that the *only error* left is the *\*variance\** of the system distortion itself, i.e. the variance in `noisy_pose` as compared to the true pose.
  - ▶ This shows our filter essentially manages to filter the noise and capture the true pose with irreconcilable 0.01 due to variance in `noisy_pose`.

# Q4. Choice of initial covariance matrices

## Process Noise Covariance Matrix (Q):

- ▶ This matrix represents the uncertainty in *change of* the system model.
- ▶ Uncertainty because of inaccurate application of control/unmodeled dynamics of environment
- ▶ Initial value is the prior on this uncertainty. We chose a diagonal matrix with diagonal values 0.1 each.

## Measurement Noise Covariance Matrix (R):

- ▶ Similar to Q, but in the measurement received. We use the values set in `trilateration.py`

## Initial State Covariance Matrix (P):

- ▶ P represents the uncertainty in the initial state of the system. Since we ourselves set the start position (to origin), we have less uncertainty and use a diagonal matrix with 0.1 as diagonal entries. We could use the zero-matrix as well.

Why all diagonal? Initially, we can assume that errors are roughly uncorrelated by supposing that they were independently generated.

## Q4: Effect of sampling time on initialization

- ▶ In the Kalman filter, the **sampling time** between corrections can influence the performance of the filter! It also influences the choice of initial covariances, as we shall see below.
- ▶ **Short Sampling Time (Higher Frequency):**
  - ▶ It is reasonable to assume that noise with shorter sampling times is smaller, because of the smaller time allowed for errors to occur. Thus we initialize the covariance to smaller values.
  - ▶ In fact, if the sensors/dynamics are noisy, frequent updates would need frequent measurements and dynamics evaluation, meaning it is hard to correct error because of this frequently accumulating sensor/dynamics error.
- ▶ **Long Sampling Time (Lower Frequency):**
  - ▶ In this case, it would be more likely for variance in errors to be larger, since there is more time for errors to add up. Thus, we would do well to choose a larger initial covariance.

## Analysis of Results Based on Sampling Time Variation

When varying the sampling time, the following key behaviors were observed:

### Fast Sampling (Shorter $\Delta t$ ):

- ▶ The filter corrected the state estimates more frequently, leading to tighter tracking of the actual robot trajectory.
- ▶ However, in the presence of noisy sensor data, the frequent corrections sometimes led to more erratic state estimates, as noise influenced the correction step too frequently.
- ▶ The MSE plot showed faster convergence initially but also higher oscillations in the error due to the influence of measurement noise.

### Slow Sampling (Longer $\Delta t$ ):

- ▶ The filter corrections were less frequent, resulting in smoother state estimates as the process model dominated for longer periods.
- ▶ However, if the model did not perfectly capture the system dynamics, larger prediction errors accumulated between correction steps, causing larger deviations from the true state.
- ▶ The MSE plot showed slower convergence and, in some cases, a larger steady-state error, particularly when the process model was less accurate.

# Thank You !

(for SC 649)