Lab 4: Particle Filter Localization

Marco Conati
Harvey Mudd College
mconati@g.hmc.edu

Evan Hassman
Harvey Mudd College
ehassman@g.hmc.edu

Bowen Jiang
Harvey Mudd College
bjiang@g.hmc.edu

***Abstract*—This document discusses the implementation of a Particle Filter with localization for Lab 4 of HMC ENGR 205. From measured data and information about our robot, we created a motion model and PF to predict the robot state at each time step as it moved in a square path around a pylon. We also implemented our filter in code, plotted our results, and calculated the path tracking error based on the true path. Overall, our filter tracked the true path closely.**

I. Introduction

Particle filters are an alternative implementation of the Bayes Filter. They are incredibly useful when dealing with nonlinear systems as this filter is able to determine the pose of a robot relative to a given map of the environment and based off of a non-linear motion model. The particle filter represents a posterior belief with a set of random state samples drawn from this posterior. At each time step, these samples can be propagated through the nonlinear dynamics and refined based on measurements to estimate a new belief. This representation is particularly useful since it is nonparametric; it can represent a broad space of distributions.

II. Particle Filter Implementation

A high level pseudo code implementation of the particle filter algorithm can be seen below.

Algorithm 1: Particle Filter

```
Result: Belief of state and set of particle states at time step $t$
1  for particle $i$ in particle set $n$ do
       // Prediction
2      pick $p_{t-1}^i$ from $P_{t-1}$
3      sample $x_t^i$ with probability $P(x_t^i|x_{t-1}^i, o_t)$
       // Correction
4      calculate $w_t^i = P(z_t|x_t^i)$
5      add $p_t^i = \begin{bmatrix} x_t^i & w_t^i \end{bmatrix}$ to $P_t^{predict}$
6  end
7  for particle $i$ in particle set $n$ do
       // Resample
8      sample $p_t^i$ from $P_t^{predict}$ with probability $w_t^i$
9      add $p_t^i$ to $P_t$
10 end
11 return $P_t$
```

The following sub-sections will discuss the implementation of these steps.

A. Motion model and PF setup

For our system, we were given initial information about coordinate systems, the measurement data, and the general mechanics of our robot.

Our global coordinate system was defined relative to cardinal directions. Positive X was due East and positive Y was due North. Our local coordinate system was given with positive X representing forward robot motion and positive Y representing leftward robot motion.

The measurement data were taken at 10 Hz. This is represented by time between samples: $\Delta T = 0.1s$. Also, we were told that the LIDAR outputted x, y values corresponding to forward and rightward in the local frame respectively.

The general mechanics of our setup were also described in the lab. The position of our landmark pylon was given as $(5.0m, -5.0m)$ in the global frame. Also, the inputs to our robot system were given as $a_{x,t}$ and $\dot{\theta}_t$ which correspond to the forward acceleration and yaw rate of the robot respectively. This gives our inputs to the system:

$$u_t = \begin{bmatrix} a_{x,t} \\ \dot{\theta}_t \end{bmatrix}$$

To get our input values $a_{x,t}$ and $\dot{\theta}_t$, we had to do some data preprocessing. First, our yaw rates were calculated by taking the differences between successive yaw measurements. Secondly, we applied a moving average filter to the acceleration data to smooth it.

With our inputs defined, we then defined our robot state. All of the robot's state variables were defined in the global coordinate frame. Our chosen state vector is as follows:

$$X_t = \begin{bmatrix} x_t \\ y_t \\ v_{x,t} \\ v_{y,t} \\ \theta_t \end{bmatrix}$$

The robot state, X_t, is calculated based on the weighted average of the state of n particles. The particles have state:

$$p_t^i = \begin{bmatrix} x_t^i \\ y_t^i \\ v_{x,t}^i \\ v_{y,t}^i \\ \theta_t^i \\ w_t^i \end{bmatrix} \quad P_t = \begin{bmatrix} p_t^1 \\ p_t^2 \\ p_t^3 \\ \vdots \\ p_t^n \end{bmatrix}$$

From our defined state, we created a motion model to describe how our state changes at each time step t based on the inputs u_t. This motion model generates global position, velocity, and yaw estimates based on yaw velocity and local robot acceleration:

$$g(\mu_{t-1}, u_t) = X_{t-1} + \begin{bmatrix} v_{x,t-1}\Delta T \\ v_{y,t-1}\Delta T \\ a_{x,t}cos(\theta_{t-1})\Delta T \\ a_{x,t}sin(\theta_{t-1})\Delta T \\ \dot{\theta}_{t-1}\Delta T \end{bmatrix}$$

Based on our sensors, we also defined a measurement vector z_t. Our robot had sensors for measuring position relative to the pylon and the robot's yaw. So, we defined our measurement vector as:

$$z_t = \begin{bmatrix} z_{x,t} \\ z_{y,t} \\ z_{\theta,t} \end{bmatrix}$$

B. Prediction Step

For the prediction step, we started with the nonlinear dynamics of our system $g(\mu_{t-1}, u_t)$, which were defined in the motion model from section II-A. For the prediction step, we must update each particles state using this motion model.

$$p_t^i = g(\mu_{t-1}, u_t) \tag{1}$$

We must also add random noise to the input u_t. This allows our particles to move in slightly different ways given a similar starting position. This represents the robot as well since the robot has error in it's ability to relate commands to exact physical output, i.e. motor rotations.

We add noise by selecting a random value from a normal distribution, $\mathcal{N}_{[0,2]}$, and adding it to the input value. The standard deviation of 2 was experimentally determined when we looked to minimize error.

C. Correction Step

The purpose of the correction step is to update the new predicted particles state measurement given sensor measurements, z_t. For each particle, the weight, w_t^i, equals the probability of the measurement given the particle, $P(z_i|x_t^i)$. The weight is normalized so that the they sum to 1.

First, we have to define a measurement model $h(\bar{\mu}_t)$ which represents our predicted measurements based on our current mean belief. To compare our actual LIDAR measurements to predicted measurements, they needed to be in the same coordinate system. However, our raw measurements are in the local robot frame and our belief state is in the global coordinate frame. To resolve this discrepancy, we decided to convert our LIDAR measurements to the global coordinate frame with the following equations:

$$\begin{bmatrix} z_{x,t} \\ z_{y,t} \\ z_{\theta,t} \end{bmatrix} = \begin{bmatrix} X_p - [y_{lidar}cos(\theta_t) + x_{lidar}sin(\theta_t)] \\ Y_p - [y_{lidar}sin(\theta_t) - x_{lidar}cos(\theta_t)] \\ \theta_t \end{bmatrix} \tag{2}$$

Where $(X_p, Y_p) = (5.0, -5.0)$ is the position of the pilon used for localization.

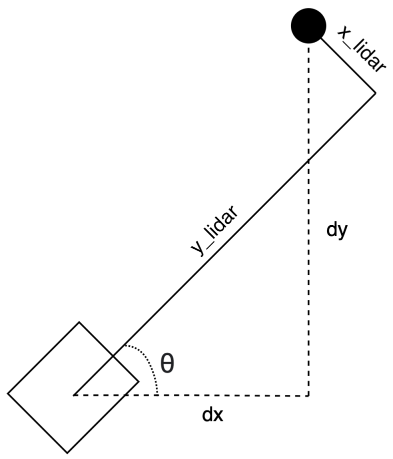

Figure 1: Lidar measurement conversion to global coordinate frame

These equations were derived from Figure 1.

This transformation makes our predicted measurement model trivial. Since our three measurements are now in the global frame and our mean belief is in the global frame, we can simply take the corresponding components of our belief state for our predicted measurements:

$$h(\bar{\mu}_t) = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$$

With the measurement model, mean, and our sensor covariance matrix, $\Sigma_{u,t} = I$, we created a multivariate normal distribution, $\mathcal{N}_{[h(\bar{\mu}_t),1]}$. Note that we used the identity for the sensor covariance matrix as the covariance is unknown to us, and we can assume no relation between measurements. With this normal PDF, we calculated $P(z_t|p_t^i)$, the relative probability of the measurement given each particle and the measurement. The relative probabilities were then normalized to sum to 1 and the particle states were corrected with these relative probabilities as the particle weights.

D. Resampling

The resampling step is a reselection of particles. This step is meant to stop weight collapse where the particles converge to a few number of points and the particle distribution is minimal. For our re-sampling, we decided to use the *choices()* function from the Python random library. This function returns a new state of particles, P_t, of the same size as the previous state of original particles, n, by randomly selecting particles with replacement from our original particle state using their weights.

When trying to solve the kidnapped robot problem, we used a slightly different methodology in resampling in order to best respond to the kidnapping. This will be further discussed in section III-D.

III. Results

A. Known Start Path Tracking

To initialize our PF with a known starting location, all initial particles were set with all state variables equal to zero:

$$\begin{bmatrix} x_t \\ y_t \\ v_{x,t} \\ v_{y,t} \\ \theta_t \end{bmatrix} = 0$$

The results of our PF with 10,000 particles are shown below. Trials were also run with 100, 1,000, and 3,000 particles; those results are included in section V.

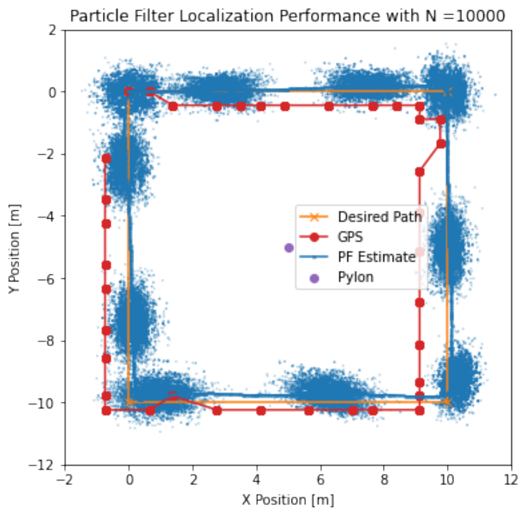

Figure 2: Known Start Path Tracking with 10,000 Particles

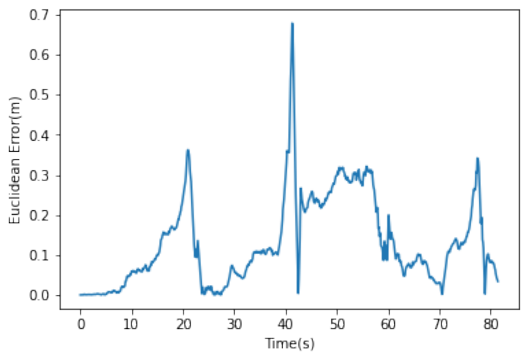

Figure 3: Known Start with 10,000 Particles Error

B. Unknown Start Path Tracking

To initialize our PF with a known starting location, all initial particles were set with randomized state variables. The positions and yaw of the particles were taken from a uniform distribution covering the entire state space. Velocities were taken from a normal distribution centered at zero with variance 1:

$$\begin{bmatrix} x_t \\ y_t \\ v_{x,t} \\ v_{y,t} \\ \theta_t \end{bmatrix} \textit{sampled from} \begin{bmatrix} \mathcal{U}_{[0,10]} \\ \mathcal{U}_{[0,-10]} \\ \mathcal{N}_{[0,1]} \\ \mathcal{N}_{[0,1]} \\ \mathcal{U}_{[0,2\pi]} \end{bmatrix}$$

The results of our PF with 10,000 particles are shown below. Similarly to the Known Start problem, trials were also run with 100, 1,000, and 3,000 particles; those results are included in the appendix.

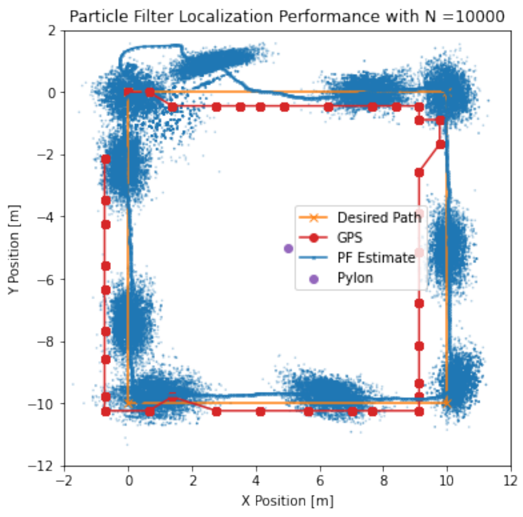

Figure 4: Unknown Start Path Tracking with 10,000 Particles

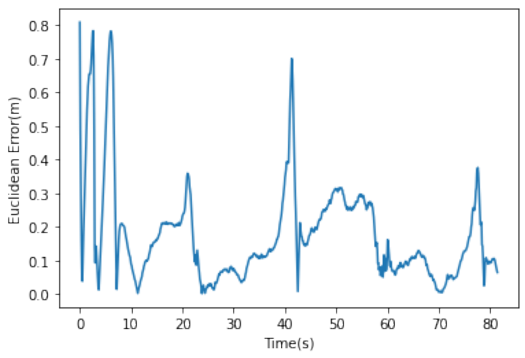

Figure 5: Unknown Start with 10,000 Particles Error

C. Comparison

Overall, the PF is able to track the desired path in both situations. Additionally, in both cases, increasing the number of particles decreased the error(with diminishing returns). However, accurate tracking begins much sooner in the known start problem. To compare performance in the two problems, we ran each for 10 iterations. The results are shown below in table I.

Known vs. Unknown Start Performance		
	Known Start	Unknown Start
Average Mean Error [m]	0.129	0.173
Average Max Error [m]	0.686	0.914
Average Variance of Error [m^2]	$4.71 * 10^{-4}$	$3.45 * 10^{-2}$

Table I: Starting State Performance Comparison

D. *Kidnapped Robot Problem*

To attempt the kidnapped robot problem, we modified our dataset. The original dataset had 815 measurements and represented the robot driving in a complete square. For this problem, we removed all measurements between index 225 and 425, which represented the southward portion of the robot's journey. So, upon completing the first segment of the square, the robot would suddenly jump to the third segment. Using our PF algorithm with a known start, we had the following results:

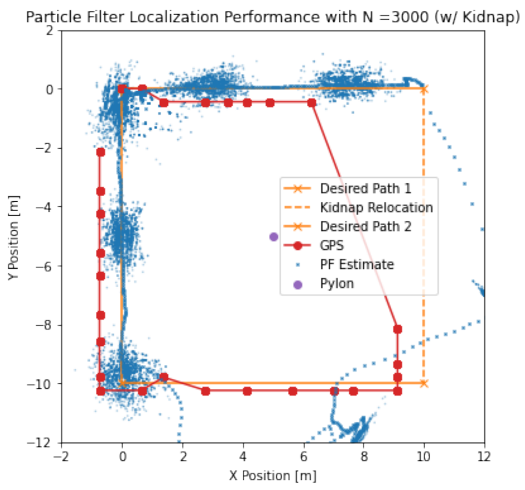

Figure 6: Kidnapped robot problem path with standard PF algorithm

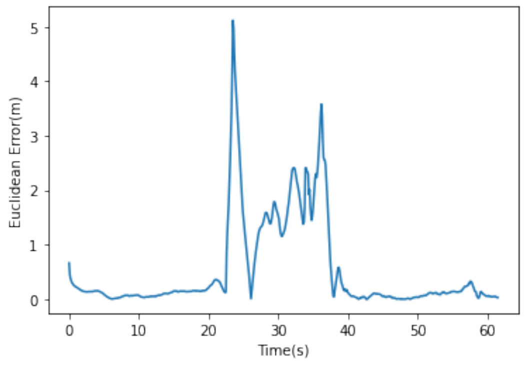

Figure 7: Kidnapped robot problem error with standard PF algorithm

The algorithm eventually rejoined the correct path, but its performance was not satisfactory. The estimated path diverged from the correct location for approximately 20 seconds after the robot was moved. To fix this error, we modified the particle filter algorithm slightly. Our new algorithm was as follows, where step two is the main modification.

1) Propagate N particles through motion model
2) Add N/10 particles sampled from a uniform distribution over the entire state space to the particle set
3) Use the measurement to calculate weights for each particle p_t^i with $w_t^i = P(z_t|p_t^i)$
4) Resample N particles from the new set where each partice has probability w_t^i

In step two, the new algorithm introduced noisy particles from across the sample space. These noise particles are generally resampled out in step four and have little effect on the position estimate during normal conditions. However, when the robot is kidnapped, noise particles near the new robot location are sampled strongly during resampling, allowing the PF to adjust to the new state quickly. Results with this new modification are shown below; the modified PF algorithm tracks the robot state and quickly adjust after it is kidnapped.

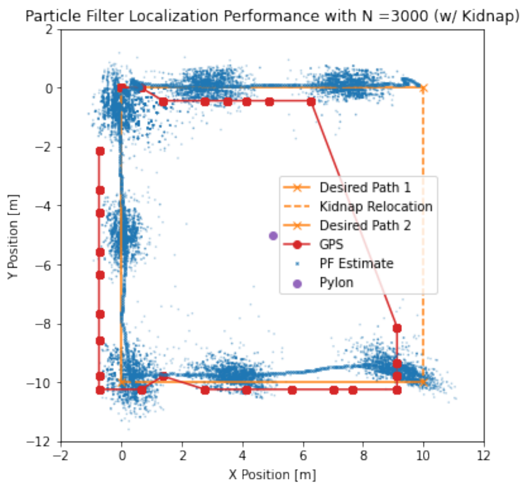

Figure 8: Kidnapped robot problem path with modified PF algorithm

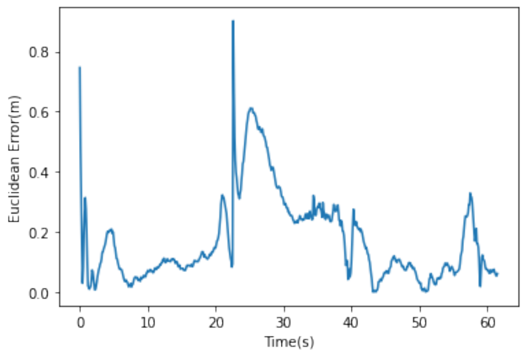

Figure 9: Kidnapped robot problem error with modified PF algorithm

IV. Conclusion

Through this lab, we gained a deeper understanding of the Particle Filter. Developing the mathematical model and moving through the implementation allowed us to gain insight in how to apply the PF to a physical use-case.

There are couple areas open for further research. Firstly, we could run deeper analysis on our number of particles. We experimented with a few different particle numbers and generally found that more particles improved performance and increased runtime. Further research could include modeling performance and runtime as a function of particle number to optimize this tradeoff.

Secondly, we could develop a more accurate sensor model. For our sensor model, we assumed that the identity matrix could be used our covariance matrix since we did not stationary sensor data. If we were able to collect stationary sensor data, we could adjust the sensor model to more accurately reflect the true sensor performance.

Without these above changes, we were still able to get fairly good results. We can see that our filter tracks the desired path closer than the GPS has the resolution to detect. Additionally, the filter was able to solve all three given problems: known initial state, unknown initial state, and kidnapped robot problem.

V. Appendix

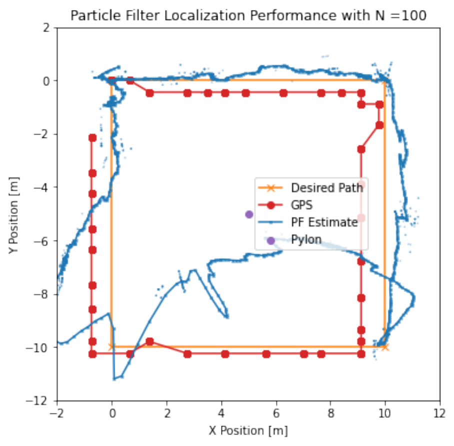

Figure 10: Known Start Path Tracking with 100 Particles

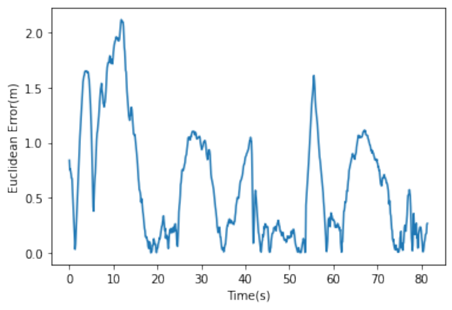

Figure 13: Unknown Start with 100 Particles Error

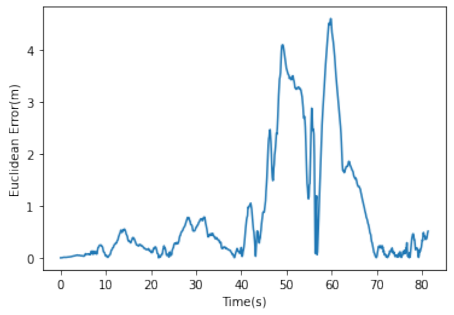

Figure 11: Known Start with 100 Particles Error

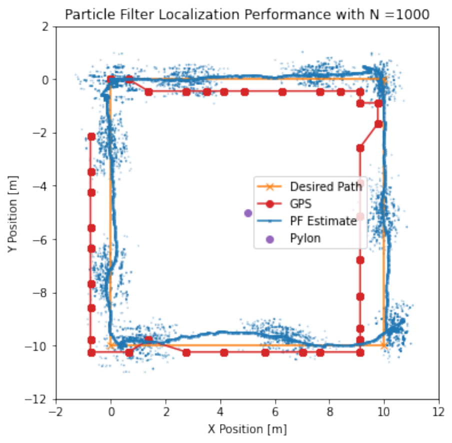

Figure 14: Known Start Path Tracking with 1,000 Particles

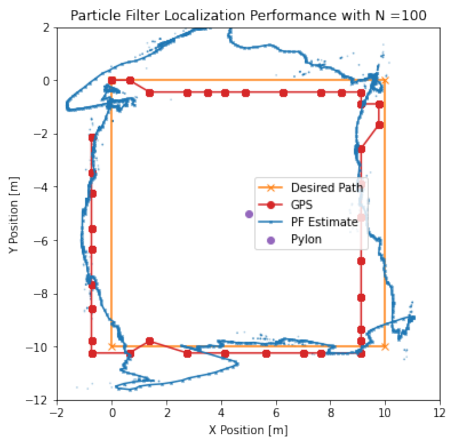

Figure 12: Unknown Start Path Tracking with 100 Particles

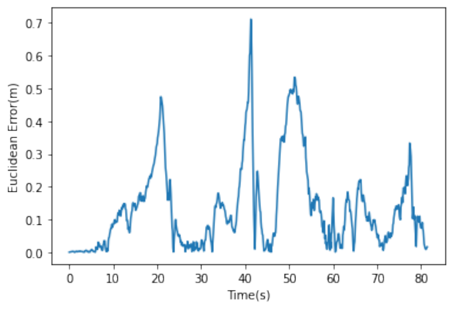

Figure 15: Known Start with 1,000 Particles Error

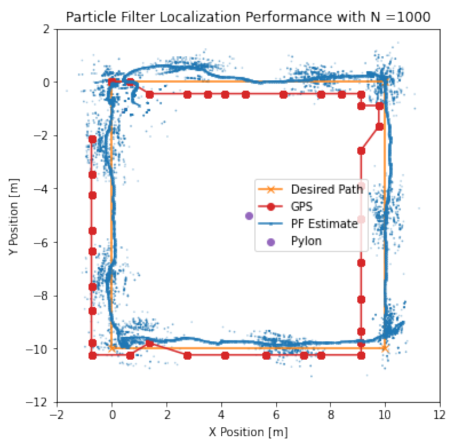

Figure 16: Unknown Start Path Tracking with 1,000 Particles

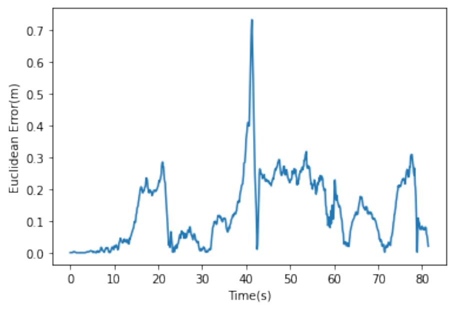

Figure 19: Known Start with 3,000 Particles Error

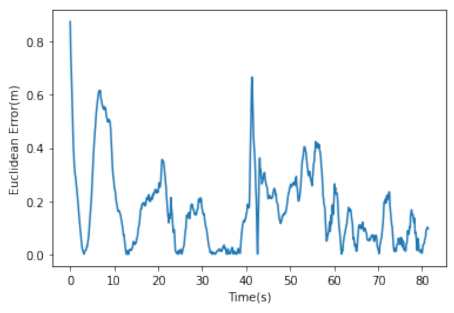

Figure 17: Unknown Start with 1,000 Particles Error

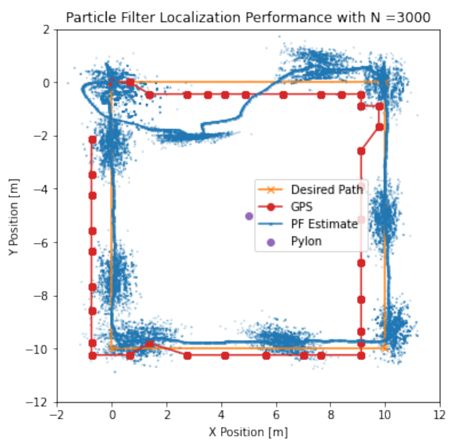

Figure 20: Unknown Start Path Tracking with 3,000 Particles

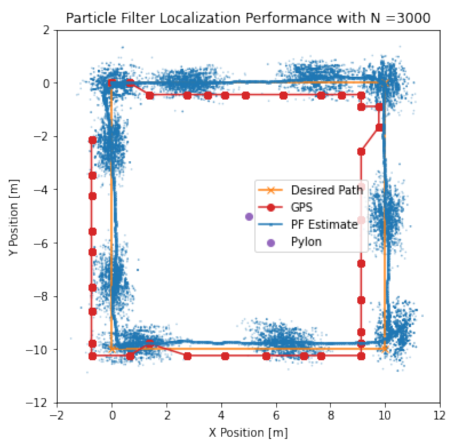

Figure 18: Known Start Path Tracking with 3,000 Particles

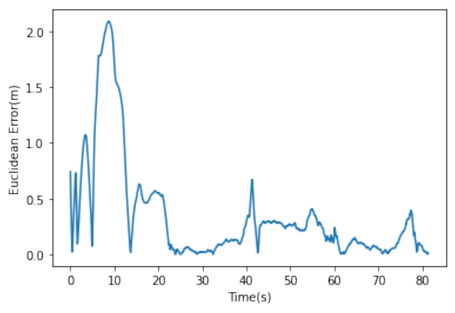

Figure 21: Unknown Start with 3,000 Particles Error

REFERENCES

[1] Shia, Victor., 31 August 2021. E205 State Estimation. Harvey Mudd College, Claremont, California.