



---

## Particle Filtering for 2D Localisation

---

October 14<sup>th</sup>, 2019

Maurice Rahme

# 1 Part A

## 1.1 Designing the Motion Model

The subject of the particle filter is a mobile robot moving on a 2D plane. Access is not given to this robot's low-level controls (i.e differential drive actuation), so it is reasonable to assume a unicycle model whereby motion can be achieved by a combination of linear and angular velocity, thus mapping the robot as a 3 degree of freedom system represented by coordinates  $x$ ,  $y$ , and  $\theta$ . Hence, the motion model used for this filter is a nonlinear model mapping  $x_t$ ,  $y_t$ , and  $\theta_t$  as follows for each next iteration of time elapsed  $dt$ :

1. If  $w$  is zero:

$$x_t = x_{t-1} + v * \cos(\theta_{t-1}) * dt + \epsilon_x \quad (1.1)$$

$$y_t = y_{t-1} + v * \sin(\theta_{t-1}) * dt + \epsilon_y \quad (1.2)$$

2. If  $w$  is nonzero, we represent motion as a circular trajectory:

$$x_{t+} = x_{t-1} + \frac{v}{w} * (-\sin(\theta_{t-1}) + \sin(\theta_{t-1} + w * dt)) + \epsilon_x \quad (1.3)$$

$$y_{t+} = y_{t-1} + \frac{v}{w} * (\cos(\theta_{t-1}) - \cos(\theta_{t-1} + w * dt)) + \epsilon_y \quad (1.4)$$

$$\theta_t = \theta_{t-1} + w * dt + \epsilon_\theta \quad (1.5)$$

where  $v$  is linear velocity,  $\omega$  is angular velocity, and  $\epsilon$  is noise.

The model is nonlinear due to the  $\sin$  and  $\cos$  relationship between the robot coordinates and velocity [1].

## 1.2 Testing the Motion Model

In this basic test, the motion model updates the robot's state using the given commands. The resultant state is appended to a Path list which is used to plot the resultant trajectory in figure 1. This plot does not showcase the motion model's nonlinearity due to the decoupling of linear and angular velocity commands.

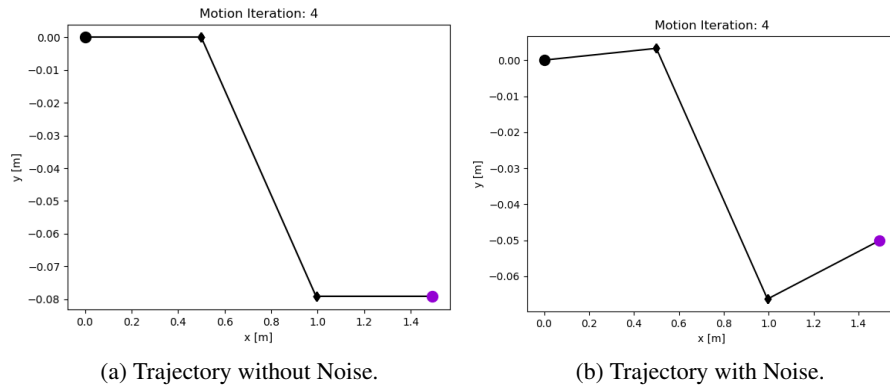


Figure 1: Testing the Motion Model.

### 1.3 Testing the Simulated Controller on the Robot Dataset

Here, the controls of the Odometry file in Data Set 0 were used to actuate the robot and plot its motion according to localisation by dead-reckoning. The data set does not provide values for  $dt$ , but it is easily extracted from the actuation time stamps, where  $dt = t_{next} - t_{current}$ . To better understand the localisation results, these were plotted against the values from the Ground Truth file provided by a camera in the robot's workspace in figure 2.

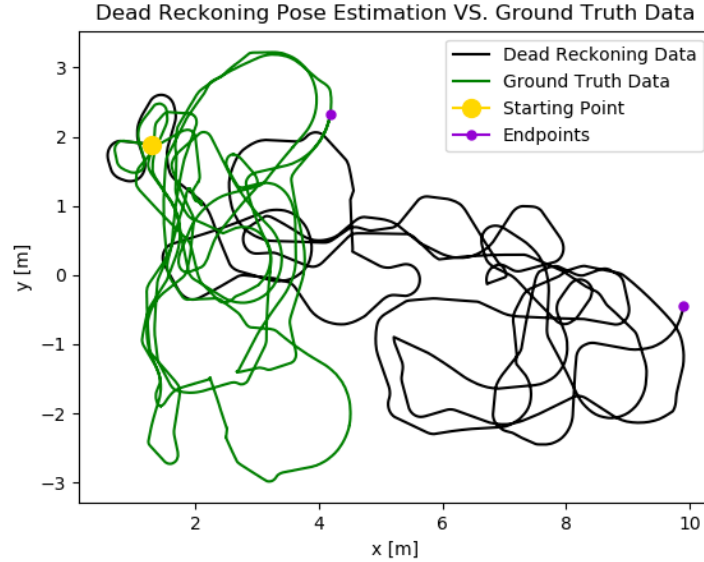


Figure 2: Dead Reckoning Pose Estimation VS Ground Truth Data, Full Set.

It is clear from this plot that the dead reckoning path diverges significantly from the ground truth path. To better identify this divergence, a plot limited at 2000 iterations can be printed, as shown in figure 3.

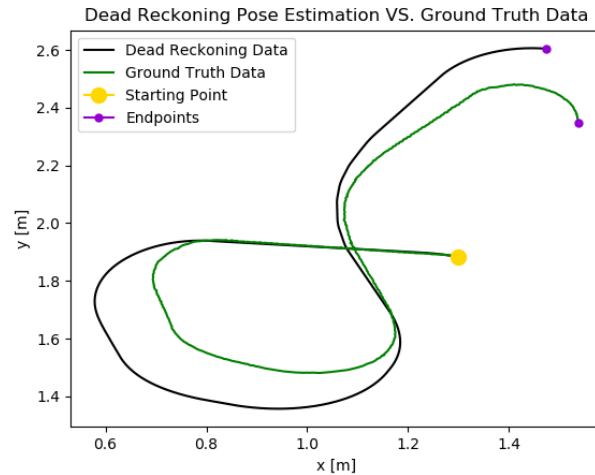


Figure 3: Dead Reckoning Pose Estimation VS Ground Truth Data, 2000 Iterations.

Examining this, it is clear that the main source of discrepancy and error arises from the issuance of  $\omega$  commands, as the dead reckoning data follows ground truth well during purely transnational motion, even recovering an initial offset due to a combination of rotational and transnational actuation done earlier.

#### 1.4 Designing The Particle Filter

The Particle Filter is a nonparametric implementation of a Bayes filter, meaning that it computes the posterior  $bel(x_t)$  of each state without needing to fit its data to a normal distribution. Instead, random state samples (particles) approximate this distribution. This filter is robust to occlusion, nonlinear dynamics, non-Gaussian noise, and most notably, it can represent multimodal posteriors, although this aspect of the filter is not needed here [2]. The filter algorithm is summarised in the following steps:

1. Generate Initial Particle Set according to the likelihood for a hypothesis to be included in set  $\chi_t$ , which is proportional to posterior  $bel(x_t)$ :  $x_t^{[m]} = p(x_t|z_{1:t}, u_{1:t})$ , where  $z$  and  $u$  are measurements and controls respectively [2].
  - (a) Knowing the robot's initial condition, generate  $M$  particles normally distributed around said position. Set the weight of each particle equal to  $\frac{1}{M}$ .
  - (b) A tunable parameter here is the standard deviation in the coordinates of the generated particles.
2. Propagate each particle forward using the motion model described in section 1.1.
  - (a) A tunable parameter here is control noise  $\epsilon$  for  $w$  and  $v$ .
3. Perform weight update if measurement is recorded.
  - (a) If a landmark measurement is recorded at any time interval between the current time stamp and the next (indicating that a measurement has been recorded during the control step), calculate the expected range and bearing for each particle using equations 1.8 and 1.9. The smaller the difference between the recorded and predicted measurements for a given particle, the better that particle acts as a representation of true state. Hence, the weight assigned is the average of  $p_{range}^i$  and  $p_{bearing}^i$  is described by equation 1.6 where  $r/b$  denotes the range or bearing estimated by the particle position,  $\mu_{r/b}$  is the actual measure range or bearing, and  $\sigma^2$  is the variance of the previous set, initialised to 1. If multiple measurements occur at a given time frame, average the weights assigned by each of them before normalising [2].
$$p(r/b, \mu_{r/b}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(r/b - \mu_{r/b})^2}{2\sigma^2} \quad (1.6)$$
  - (b) Normalise the weights of all particles by dividing each by the sum of all the weights so that they add up to 1. Note that in every iteration, weight by multiplying the weight of a given particle at the previous iteration with the weight generated in this step [2].
4. Calculate  $w_{var}$ , the variance of the set's weights. A value tending to zero indicates that that all weights are converging to the same value, discouraging the use of a sampling step. Conversely, a high variance encourages resampling to redistribute the weights among weaker particles.  $w_{var}$  is given by:

$$w_{var} = \frac{\sum (w_t^i - w_\mu)^2}{M - 1} \quad (1.7)$$

5. If  $w_{var}$  is above  $4E - 10$ , resample using low-variance resampling, which discards low-weight particles and replaces them with randomly generated ones.
  - (a) Through a sequential stochastic process, the algorithm chooses a random number  $r$ , then - to build the new set - selects particles according to the weight-space division of  $u = r + (m - 1) * M^{-1}$  as shown in figure 4 [2].

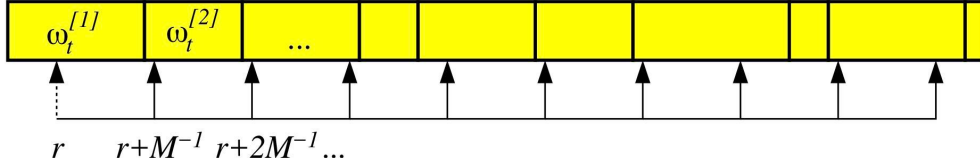


Figure 4: Low-Variance Resampling Weight-Space Division [2].

- (b) Within each division, the particle of index  $i$  selected is the one that corresponds to  $c < u$ , where  $c = c + w_t^i$ . Until this is true,  $i$  is incremented to evaluate the weight of the next particle in the division. Once the division is broken by  $c < u$ , the particle is added to the set and the next division is examined [2].
6. If  $w_{var}$  is below  $0.2E - 10$ , in the resampling step, generate  $0.1 * M$  particles normally distributed around the prior to restore variance to the set.
7. Extract the posterior from the sample set by computing a weighted average of  $x$ ,  $y$ , and  $\theta$  separately using the particle weights. The variance is extracted here for the next iteration using equation 1.7.
8. Repeat steps 2 through 7 for the entirety of the robot's motion.

A key design consideration is the number of particles,  $M$ . The particle filter is designed with a theoretical infinite sample set  $\chi_t$ . The closer  $M$  tends to one, the less the measurement probability contributes to the weighing of particles due to an introduction of systematic bias to the posterior. This is evident in the extreme case of  $M = 1$  where the single particle will always be taken as the posterior irrespective of its weight. However, increasing  $M$  increases computational power, so an intermediate value of 1000 was chosen to compromise between speed and performance. Increasing  $M$  also increases variance [2].

Another important factor is sample variance. This is especially true for Data Set 0, where the first measurement occurs only after 382 motion iterations, by which time the particles can easily converge to a single point due to the lack of measurements. This is the reason for which  $w_{var}$  is used to determine resampling, as well as the introduction of new random particles to the set as described in the algorithm [2].

## 1.5 Specifying the Measurement Model

The measurement model maps the range and bearing - as measured by the robot at various timestamps - to the cartesian position of the measured object (the landmark) relative to the robot, and vice versa. Notably, for the particle filter, given a landmark - whose cartesian position is known - the measurement model is used to compute the expected range and bearing of each particle in the active set. These expected readings are then compared to the actual reported sample, and are then used to assign a weight to each particle as described in section 1.4.

The model for expected range and bearing given known landmark and particle coordinates is given as follows [3]:

$$range = \sqrt{(x_t - x_i - \epsilon_{xi})^2 + (y_t - y_i - \epsilon_{yi})^2} \quad (1.8)$$

$$bearing = \arctan\left(\frac{y_i + \epsilon_{yi} - y_t}{x_i + \epsilon_{xi} - x_t}\right) \quad (1.9)$$

Where  $x_t, y_t$  denote the robot's cartesian coordinates,  $x_i, y_i$  denote the landmark's cartesian coordinates,  $\epsilon_{x_i, y_i}$  is standard deviation of each landmark as reported by the Vicon camera, and  $\epsilon_{x_t, y_t}$  is the robot's sensor noise, which was arbitrarily selected as 0.0001 as it is assumed to be worse than the Vicon camera, and no steady samples were provided to analytically compute noise.

## 1.6 Testing the Measurement Model

A simple test was conducted to assess the validity of the measurement model, whereby equations 1.8 and 1.9 were used to compute range and bearing respectively for known robot and landmark positions. These values were then fed into the complement of the measurement model, which computes the landmark position based on the robot's coordinates and the range and bearing computed by equations 1.8 and 1.9. The complement of the measurement model is described below:

$$x_i = x_t + range(\cos(bearing + heading)) \quad (1.10)$$

$$y_i = y_t + range(\sin(bearing + heading)) \quad (1.11)$$

Where *heading* is simply the robot's orientation relative to the domain's  $x$  axis. Below are the results from running the test with and without measurement noise:

```

Add noise to the measurement model? [y/n]n
Measurement 1:
Error in x[m] and y[m]: [6.106226635438361e-16, 0.0]
Range[m]: 8.09393468689
Bearing[m]: -1.75882673717

Measurement 2:
Error in x[m] and y[m]: [2.220446049250313e-16, 1.1102230246251565e-16]
Range[m]: 2.5728931992
Bearing[m]: -1.20610126119

Measurement 3:
Error in x[m] and y[m]: [0.0, 4.440892098500626e-16]
Range[m]: 5.06332745234
Bearing[m]: 1.10650362626

Note that a nonzero, but infinitesimal (E-16) value may be returned due to the rounding of numpy.trigonometry functions

```

(a) Test without Noise.

```

The added noise in x and y is 2.53129793432e-09 and 2.66422454249e-08
The added noise in x and y is 2.54970966633e-09 and 2.56402644498e-12
The added noise in x and y is 4.07128580151e-07 and 7.81588431905e-07
Error in x and y for measurement 1: [7.430208281977713e-09, 2.571009005691849e-08]
Error in x and y for measurement 2: [1.902723867353018e-09, 1.6972535599180105e-09]
Error in x and y for measurement 3: [3.820185368041962e-07, 7.941637094610599e-07]

```

(b) Test with Noise.

Figure 5: Testing the Measurement Model.

## 2 Part B

### 2.1 Assessing Filter Performance

#### 2.1.1 Performance in the Basic Motion Test

If noise is added to the motion model for the commands in step 2, it deviates slightly from the expected trajectory. Conversely, as seen in figure 6, the particle filter traces a close trajectory even with the absence of measurements. However, this is not true for every instance, as it is sometimes less accurate than the dead reckoning path depending on the inherent randomness introduced by the particle set, which is not corrected by measurements in this case.

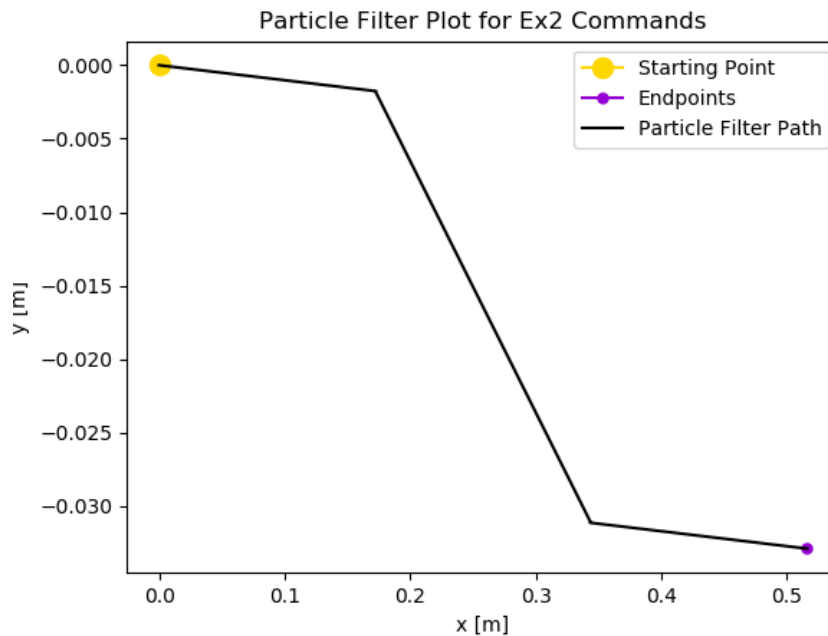


Figure 6: Particle Filter Trajectory for Exercise 2 Commands (With Noise).

### 2.1.2 Performance in the Odometry Data Test

Compared to dead reckoning as shown in figure , the filter follows the ground truth path more closely. Note that the dead reckoning path behaved well in cases of near-total or total linear motion, but deviated significantly with the introduction of angular motion. Hence, in the initial particle distribution, the standard deviation for the  $x$  and  $y$  coordinates of the particles was kept low at 0.0002, but that of  $\theta$  was kept comparatively higher at 1.5 so that particles would be able to explore a larger sample space of  $\theta$  possibilities, as this was the main source of error. The particle filter path stays within the general area of the ground truth path, as seen in figure , an improvement on dead reckoning, which is completely out of bounds. The Legend was omitted here to make space for the plot.

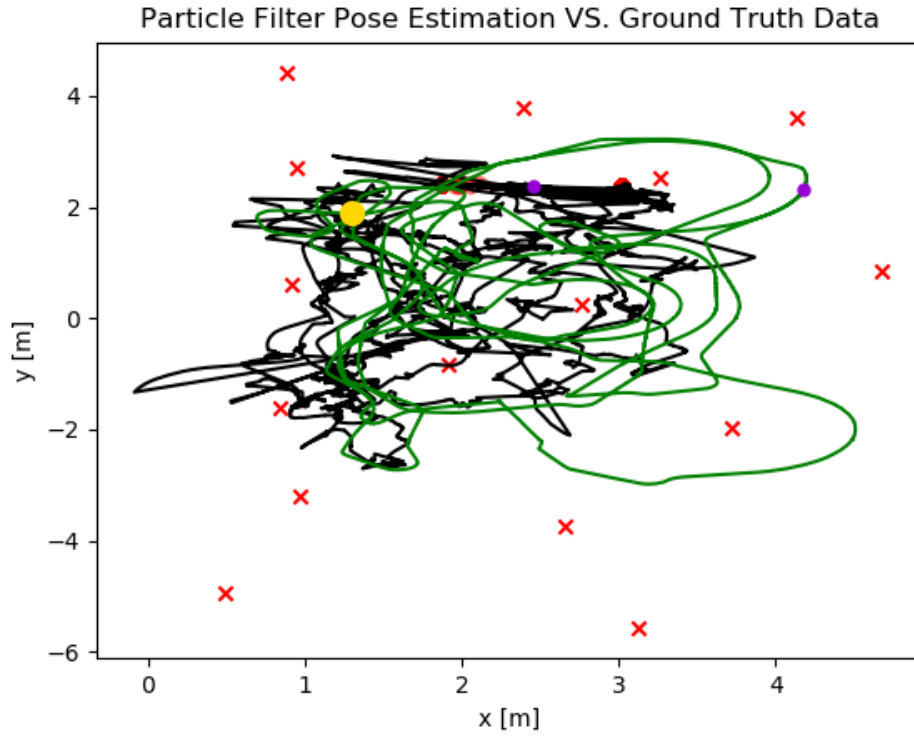


Figure 7: Particle Filter VS Ground Truth Path.

However, as can be seen by figure , the particle filter path still deviates significantly from the ground truth path. Its detailed behaviour is explored in the next section, but one significant contribution is be the weight update operation using the previous state variance in equation 1.6, as changing this value has an effect on behaviour.

Noting that the weight update is performed using a measurement, clear attempts to correct at these time stamps can be identified by the jagged nature of the particle filter path in figure . This means that an increase in measurement frequency - there are approximately 7,719 measurements for a set of 95,817 motion updates - could result in a smoother and more accurate path.



## 2.2 Uncertainty in the Algorithm

The filter produced here is effective, but is in need of further tuning for the task at hand. There are many parameters to optimise, including motion model noise, sensor noise, the  $w_{var}$  threshold for resampling, the standard deviation of the initial normally distributed sample set, and the sample set size. With this variety of optimisation routes, it was difficult to identify a clear design procedure. However, certain elements showed more promise, notably the standard deviation of  $\theta_t$  around the prior in both the initially generated set and the random sample replacements in the resampling stage. A higher comparative value of 1.5 to 2, depending on other parameters, showed better performance as it geared the particle filter to evaluate particles mainly using their bearing measurement, hence corrected the rotational motion deviation mentioned in section 1.3. Below are plots for  $\theta$  of 1.5 and 2 respectively.

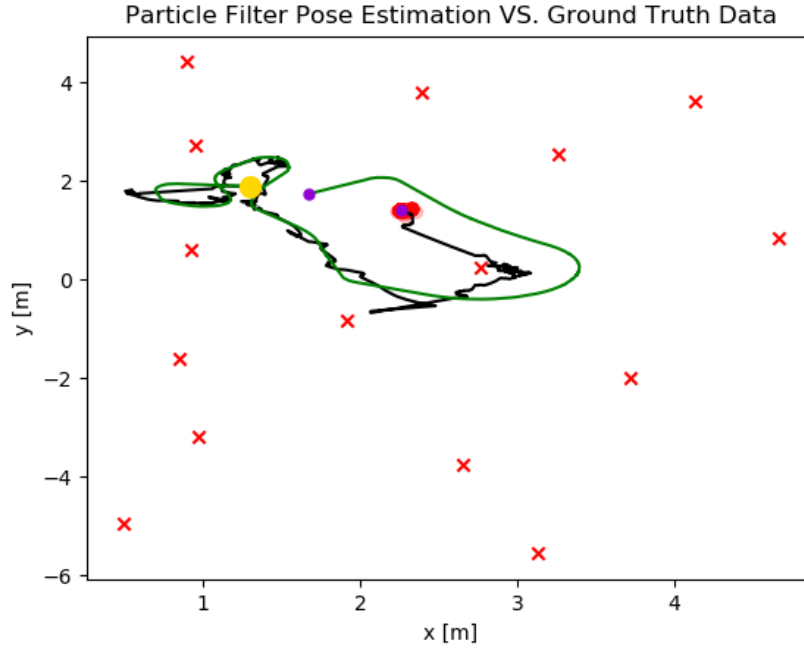


Figure 8: Particle Filter for  $\theta = 1.5$ .

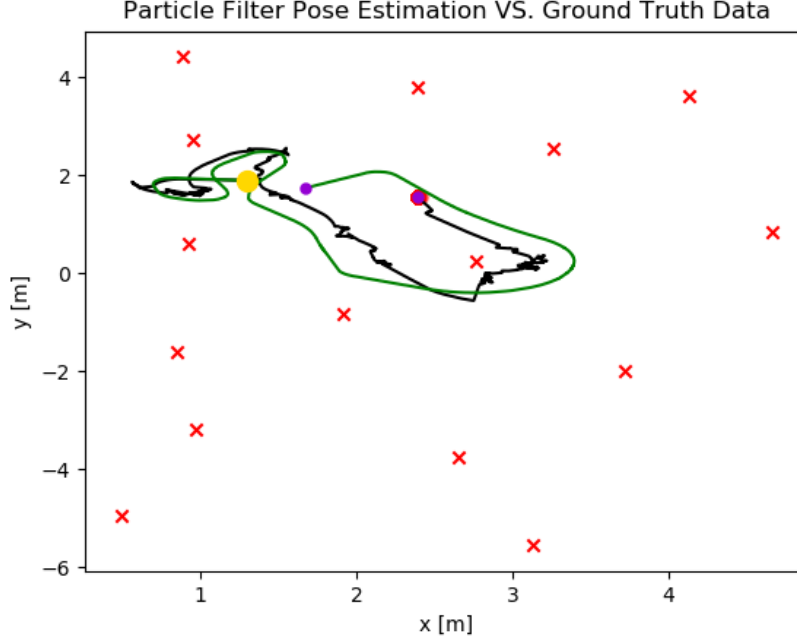


Figure 9: Particle Filter for  $\theta = 2$ .

Figure shows better rotational performance than figure , but the latter shows better linear performance.

Another important variable was found to be  $w_{var}$ , as this determined the resampling frequency. The chosen minimum of  $4E - 10$  for resampling and maximum of  $2E - 10$  for introducing random particles into the resampling were chosen experimentally, but could be further optimised by examining the outcome of resampling and particle introduction frequency over various portions of the control set. In tandem, the number of random particles to be generated in the special resampling step is another tunable parameter with a significant effect on performance.

Crucially, the filter performed somewhat poorly - especially over time - in the absence of measurements. An attempt was made to expand the measurement set, but the validity of measurements included outside of the control actuation range was difficult to assess, as they varied in temporal distance, relevance and importance across the data set. The plot below shows significantly poorer performance when no measurements are taken, which puts into perspective the first 382 filter iterations, for which there is no measurement and hence no weight update. This means that the first 382 iterations - noting that the filter could still be improved - sometimes cause the path to tend towards an un-recoverable state. This can be seen by running the code multiple times without changing any parameters.

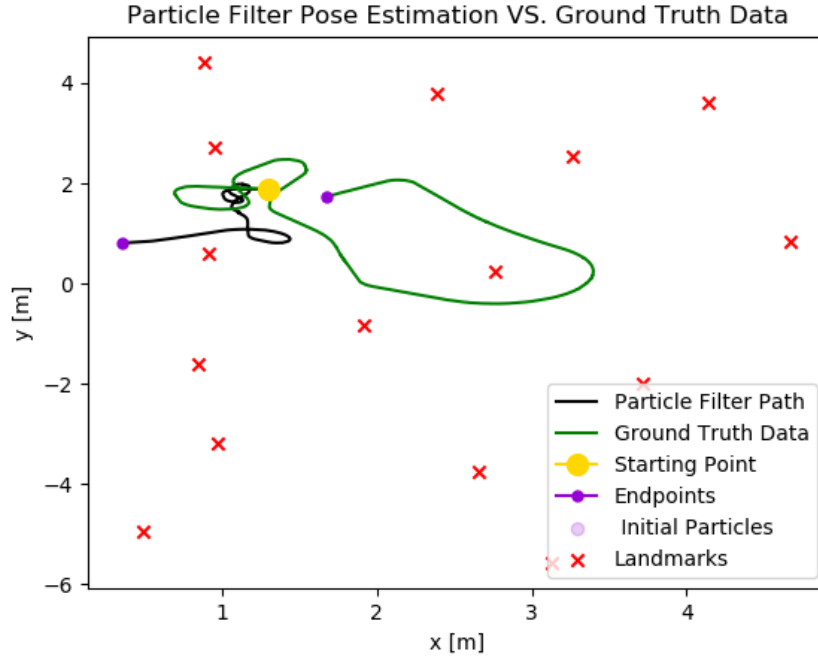


Figure 10: Particle Filter with no measurements.

Notably, this trajectory is much smoother than those in figures 2.2 and 2.2. However, it is evidently a worse performer, highlighting the importance of the measurement update and the destabilising effect caused by its absence in the beginning of the control sequence. The later measurements are implemented, the longer the filter will take to recover. This is especially true if measurements are scarce.

### 3 Citations

- [1] “Robot Motion.” Probabilistic Robotics, by Sebastian Thrun et al., MIT Press, 2010.
- [2] “The Particle Filter.” Probabilistic Robotics, by Sebastian Thrun et al., MIT Press, 2010.
- [3] Russel, Stuart Norvig Peter. Artificial Intelligence: a Modern Approach. PEARSON, 2018.