# Particle Filter Localization

Autonomous Robot Navigation (E160) Lab 4 Report

Sean Mahre
*Department of Engineering*
*Harvey Mudd College*
Claremont, CA
smahre@hmc.edu

Eyassu Shimelis
*Department of Engineering*
*Harvey Mudd College*
Claremont, CA
eshimelis@hmc.edu

***Abstract*—This report details the design and implementation of a particle filter in a differential drive robot. The filter must accurately localize the agent in both known and unknown locations. In simulation, a particle filter with 150 particles estimated the robot state with a root-mean-square error (RMSE) of $0.131\text{m}$. In hardware, the lack of a ground truth meant that we could not quantify the localization error. The point-tracker was used to close the point tracker feedback loop, and the localization results remained consistent despite growing odometry errors.**

## I. Introduction

Localization under uncertainty has been an important breakthrough in the field of robotics in the past few decades. In the previous lab, only odometry was used to estimate the robot's state; however, this type of localization is prone to accumulating error over time.

## II. Background

### A. Particle Filtering

The particle filter is a nonparametric filtering technique used to localize agents in noisy or uncertain environments. The filter relies on an approximated posterior, rather than a posterior with a fixed functional form, such as a Gaussian [1].

Particle filters maintain a discrete number of belief states in a large state space. This discretization allows for a fast posterior approximation, at the cost of reduced accuracy. The particle filter algorithm is outlined in Alg. 1.

## III. Problem Definition

The goal of this lab is to design and implement a particle filter on an E160 robot. The filter must remain robust despite noisy IR range sensor measurements.

The E160 robot is a differential-drive robot operating in a two-dimensional space. Its state $\xi$ consists of its position and heading,

$$\xi = [x, y, \theta]^T . \tag{1}$$

We would like to approximate this state with a finite set of particles $\mathcal{X}$ at time $t$:

$$\mathcal{X}_t := \{\xi_t^m, w_t^m\}_{m=1}^M,$$

where $\xi_t^m$ is the belief state with an associated weight, $w_t^m$.

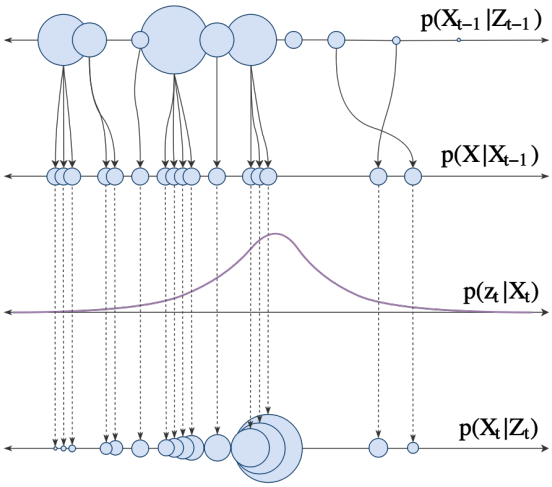



Fig. 1: A step-by-step particle filter example in a single dimension. Each circle represents a belief state; the radius being proportional to the estimate's weight.

## IV. Particle Filter Design

At each time step, every sample in the particle set $\mathcal{X}_{t-1}$ is propagated using a noisy kinematic model. Each particle is weighted by comparing its expected sensor measurements to the actual measurement, as described in Section IV-A. Finally, the weighted particles are sampled into a new set $\mathcal{X}_t$ with a selection probability equal to its relative weight. This algorithm is specified further in Alg. 1 [1].

### A. Wall Distance Calculation

Each particle is weighted based on its position in the map and its expected sensor measurements. Recall, the E160 robot has three IR range sensors: one facing forwards, and two on either side. At each time step, the weight of each particle is determined by comparing the actual measurement $z$ to the expected measurement $z_{exp}$. For each particle, the expected sensor reading must be computer from its current position in a known map. Generally, for each range sensor, the expected sensor reading is the minimum distance to every wall in the map.

**Algorithm 1** Particle Filter Algorithm

**function** PARTICLE_FILTER($\mathcal{X}_{t-1}$, $u_t$, $z_t$):
  $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ ▷ Initialize new particle set
  **for** $m = 1$ to $M$ **do**
    sample $x_t^m \sim p(x_t|u_t, x_{t-1}^m)$
    $w_t^m = p(z_t|x_t^m)$ ▷ Using probabilistic sensor model, see section IV-B
    $\bar{\mathcal{X}} = \bar{\mathcal{X}} + \langle x_t^m, w_t^m \rangle$
  **end for**
  **for** $m = 1$ to $M$ **do**
    draw $i$ with probability $\propto w_t^m$
    add $x_t^i$ to $\mathcal{X}_t$
  **end for**
**return** $\mathcal{X}_t$
**end function**

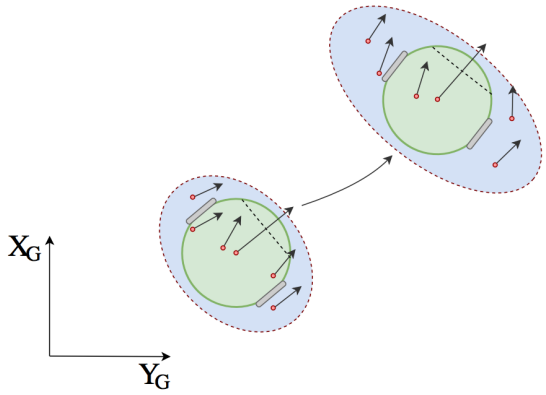



Fig. 2: Particle filter diagram for a differential drive robot. Each particle is an instantiated belief of the robot's state. At each time, each particle is propagated using a noisy kinematic model. This leads to growing uncertainty, which is corrected during resampling.

The vectors intersect if $\exists\ k, v$ such that

$$\mathbf{w} + k\mathbf{d} = \mathbf{b} + v\mathbf{h}.$$

There are two unknowns, but only a single equation. Taking the cross product of both sides by $\mathbf{d}$,

$$(\mathbf{w} \times \mathbf{d}) + k(\mathbf{d} \times \mathbf{d}) = (\mathbf{b} \times \mathbf{d}) + v(\mathbf{h} \times \mathbf{d}).$$

The cross product of any vector with itself is equal to zero, thus $\mathbf{d} \times \mathbf{d} = 0$, leaving us with a single unknown variable, $v$.

$$(\mathbf{w} \times \mathbf{d}) = (\mathbf{b} \times \mathbf{d}) + v(\mathbf{h} \times \mathbf{d}).$$

Solving for $v$,

$$v = \frac{(\mathbf{w} - \mathbf{b}) \times \mathbf{d}}{\mathbf{h} \times \mathbf{d}}.$$

The distance $z$ to the wall is then

$$z = ||v\mathbf{h}||. \tag{2}$$

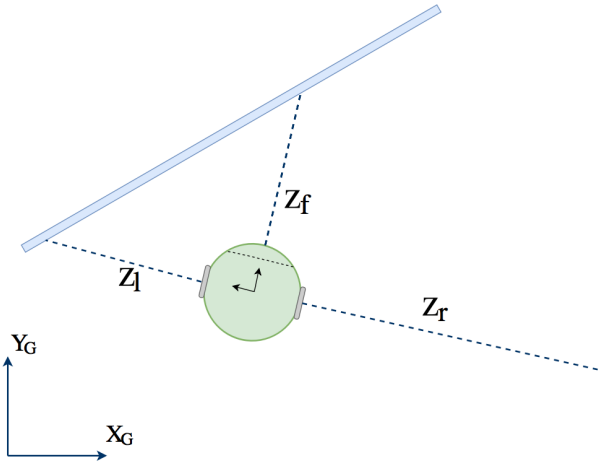



Fig. 3: The E160 robot has three IR range sensors, where each measurement $z = [z_f, z_r, z_l]$.

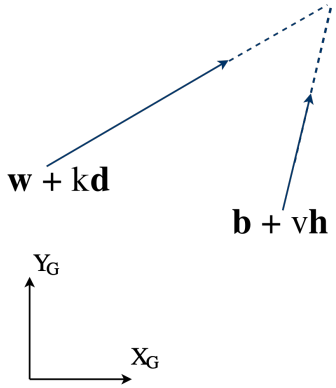



Fig. 4: Vector representation of the wall and the front range sensor of the robot. This method allows us to compute wall intersections by solving for the magnitude of the scaling terms, $k, v$.

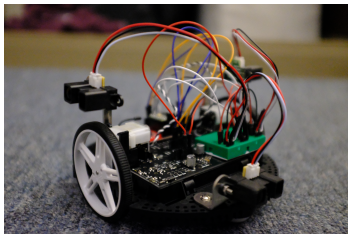
If $\mathbf{h} \times \mathbf{d} = 0$, the vectors are parallel, and no solution for $v$ exists. The sensor heading $h$ intersects with the wall if, and only if,

$$\begin{aligned} 0 \ &\leq \ k \ \leq \ 1 \\ 0 \ &< \ v. \end{aligned}$$

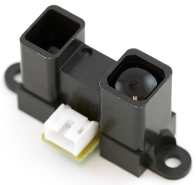
### *B. Probabilistic Sensor Model*

The E160 robot has three Sharp GP2Y0A02YK0F infrared (IR) range sensors, as pictured in Figure 5b. Each sensor has an operational range of 15 to 150cm. The sensor operates by measuring the position of a reflected beam of light on a position sensitive detector (PSD). Unfortunately, these types of sensors are subject to noise both internally and due to a dynamic environment.

An important part of developing a particle filter is creating an accurate model of the sensor noise, as this is what is used to weight each particle in the set.



(a) The chassis diameter measures 14.15 cm $\pm$ 0.01 and each wheel has a diameter of 6.95 cm $\pm$ 0.01.



(b) Sharp GP2Y0A02YK0F infrared (IR) range sensor. from 15 to 150cm

Fig. 5: The E160 robot chassis and range sensor

Range sensor measurement models $G$ are often represented as normal distributions about the true measurement $z_{exp}$ ($G \sim \mathcal{N}(z_{exp}, \sigma_s)$), where $\sigma_s$ is an experimentally determined standard deviation of the sensor's noise, see Eq. 3. The noisy measurement model is represented in the top left of Figure 6.

A more complete model, however, will combine multiple probabilistic sensor models in order to account for other factors that may affect the distribution of measurements. This model is called the beam-based sensor model; a mixture of four probabilistic sensor models [2].

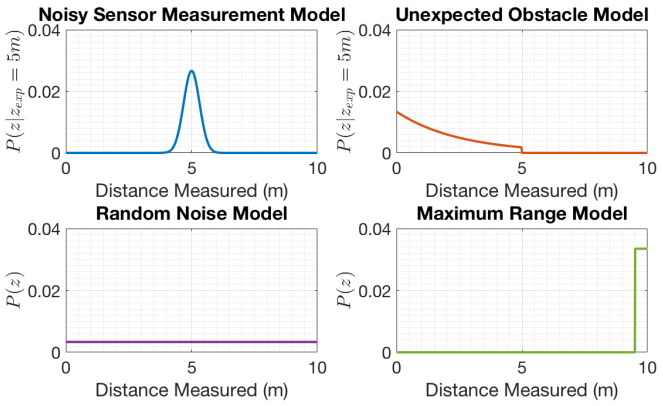



Fig. 6: The probabilistic beam sensor model is composed of four distributions, each meant to model an accurate sensor probability distribution $p(z|z_exp = r)$.

$$p_n(z|z_{exp}=r) = \eta \frac{1}{\sqrt{2\pi\sigma_s}} e^{-(z-r)/2\sigma_s}, \tag{3}$$

where $\eta$ is a normalization factor.

In addition to the noisy measurement model, an obstacle model is included to represent the probability of unexpected range measurements such as people walking around. This is represented in Eq. 4 as an exponentially decreasing function up to the robot. The unexpected obstacle model represented in the top right of Figure 6.

$$p_o(z|z_{exp}=r) = \begin{cases} \eta\lambda e^{-\lambda z}, & \text{if } z < r \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

The third element to the probabilistic sensor model is a random noise component. This random noise, as shown in Eq. 5, is uniformly distributed across all possible range measurements. The random noise model is represented in the bottom left of Figure 6.

$$p_r(z) = \eta \frac{1}{z_{max}} \tag{5}$$

The final component is the maximum range model, described in Eq. 6. The IR range sensor has a limited range, and for very large maps with distant walls, it is very likely that the sensor will be unable to measure the distance, thus reporting a maximum measurement reading. The maximum range model is represented in the bottom right of Figure 6.

$$p_m(z) = \eta \frac{1}{z_{small}} \tag{6}$$

In Eq. 7, we see that the final sensor model $G$ is a mixture of the four models. This is the sensor model that is used to weight each particle in the particle filter.

$$\begin{aligned} &G(z|z_{exp}=r) = \\ &p_n(z|z_{exp}=r) + p_o(z|z_{exp}=r) + p_r(z) + p_m(z) \end{aligned} \tag{7}$$

This model varies from sensor to sensor. The model for a specific sensor can be determined using regression, on functions 3 - 6.

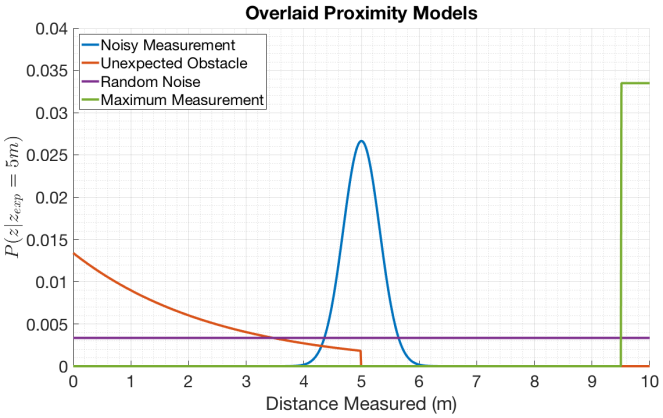



Fig. 7: Overlaying these models shows the relative importance of each component.

### C. Particle Weighting

At every iteration

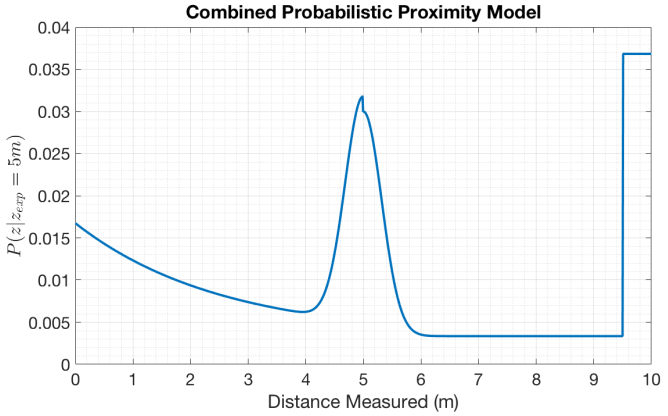



Fig. 8: The final probabilistic sensor model is a mixture of all of the separate models outlined above.

### D. Particle Resampling

A common method is to resample the particle set at every iteration of the algorithm, or after every $n$ iterations; however, a slightly more efficient algorithm will resample only when needed [3] [4]. These methods only resample when the magnitude of weights are below a predefined threshold [5]. This type of resampling technique will allow particles to propagate for longer periods of time before resampling, which alleviates the issue of particle deprivation.

## V. Filter Implementation

The E160 robot uses an XBee wireless module to wirelessly communicate with an off-board computer. All odometry and sensor measurements are transmitted from the robot to the off-board computer at each time step. The particle is implemented in Python and also runs on the external computer. The external computer localizes the robot using the particle filter, computes the required control efforts, and wirelessly transmits the control output back to the robot.

The Python program has a graphical user interface (GUI). The GUI, shown in Figure 9, displays the map, important robot states, and the particles.

At each time step, the particle filter localized the vehicle, and the control outputs were wirelessly transmitted to E160 robot.

## VI. Testing and Results

### A. Map

The particle filter was tested in simple hallway. The hallway has a footprint of approximately $4\text{m}$ by $4\text{m}$, the specific measurements are shown in Figure 10. To overcome the sensor's limited range, the robot's path was constructed next to the walls.

Note that in Figure 9, there are two sets of particles, representing two separate beliefs because of the map's symmetry. This type of behavior is desired when the robot starts at an unknown location. The two beliefs should be maintained until the robot reaches a unique point in the map; in this case, the center of the hallway.

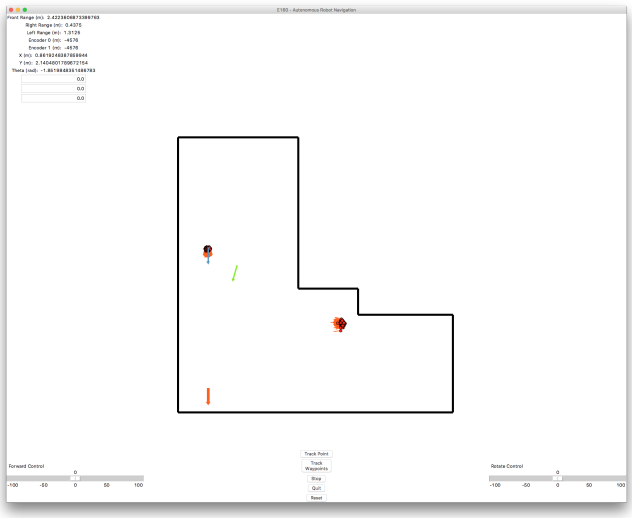


Fig. 9: The E160 graphical user interface. The GUI displays the map (walls colored black), the estimated odometry state (orange arrow), the estimated particle filter state (light-green arrow), the desired state (orange arrow), and the particles (red circles).

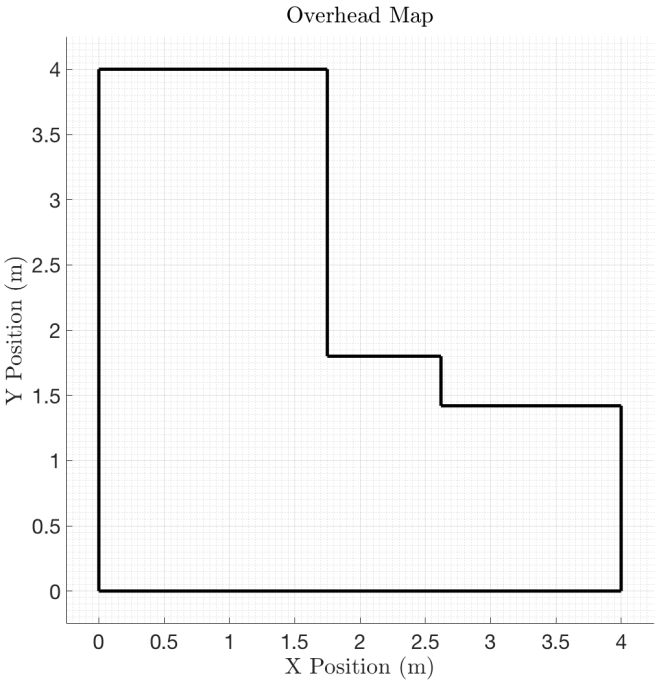



Fig. 10: The hallway map used for simulation and hardware tests. The origin is located at the bottom right corner of the hallway. Each hall has a different width: $1.75\text{m} \pm 0.01$ and $1.42\text{m} \pm 0.01$ for the vertical and horizontal halls, respectively.

### B. Simulation

The particle filter was first tested in simulation. The goal of the first test was to characterize the performance of the particle filter from a known path through a predefined path. In Figure

12, the blue path indicates the odometry-only path, and the orange is the estimated location of the robot, using a particle filter with 150 particles. In this experiment, root-mean-square error (RMSE) is 0.0315m.

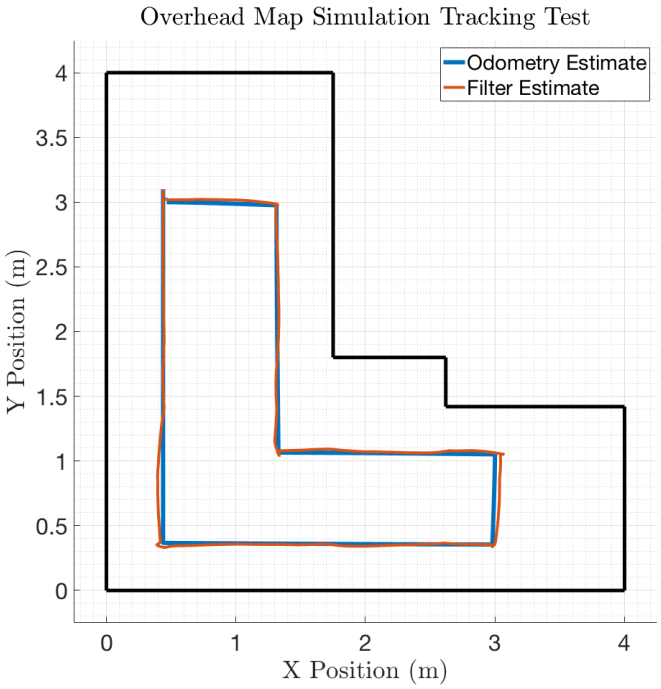



Fig. 11: Simulated particle filter test with a known start. The robot begins in the top left corner of the map and traverses in a counter-clockwise direction, remaining close to the walls of the hallway.

The next experiment was to ensure that the particle filter could converge when with an unknown start. For this experiment, the entire map is populated with 1000 particles, that slowly converge to the true state. An example of this convergence is illustrated in Figure 12. Out of 10 random-start experiments, the particle filter converged to the true position in 8 of them.

To summarize, the performance of the particle filter in software is illustrated in Table I.

| | **Tracking Test** | **Convergence Test** |
|---|---|---|
| **RMSE** (m) | 0.0315 | - |
| **Convergence Rate** | - | 0.8 |

TABLE I: Results of the simulation tests.

### *C. Hardware*

A similar tracking test was performed in hardware. The robot was placed at a known start and tracked a similar path around the hallway, using the odometry estimate as the true position. The results are illustrated in Figure 13. This resulted in a RMSE of 0.1388m, over four times as worse as the simulation.

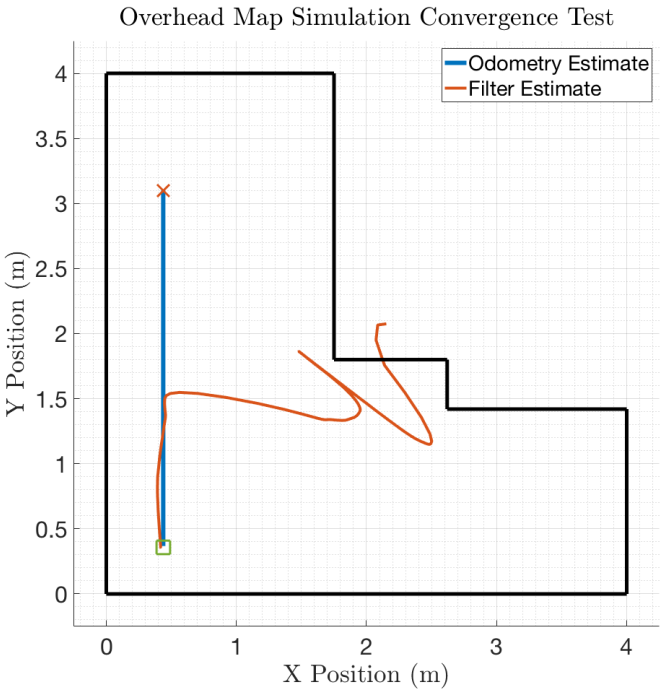



Fig. 12: The robot begins at an unknown location, the red x, and drives towards the green square. The filter estimate in orange is stuck between both hallways until finally converging to the true state when the robot enters a significantly unique point in the map.

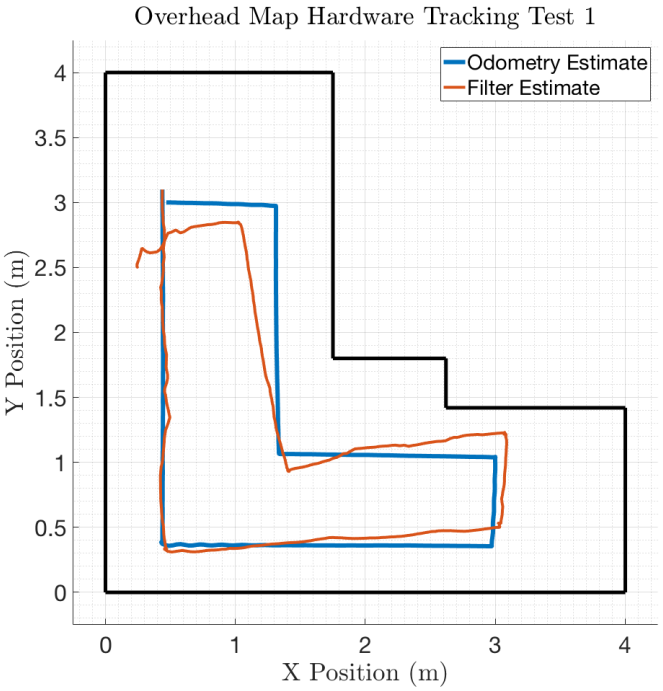



Fig. 13: Hardware particle filter test with a known start. Notice that the filter estimate worsens with time, this is because the odometry estimation is assumed to be the true state.

Unfortunately, there is no ground truth to compare the particle filter results. The best approximation is the odometry results; however, the error is unbounded.

One method to test this is to use the particle filter output as the state estimate in the robot's point tracker. This will test two things: first, that the controller remains stable; and second, that the robot's path does not drift with time. Ideally, we should expect the particle filter estimate to maintain a consistent path around the hallway. This was tested by running the robot through three laps around the map, with the particle filter used for localization.

In the first feedback experiment, the robot was placed at the starting position and the robot was driven multiple times around the predefined path. This experiment is designed to show the error that accumulates in the odometry, despite an accurate start. The result is illustrated in Figure 14.

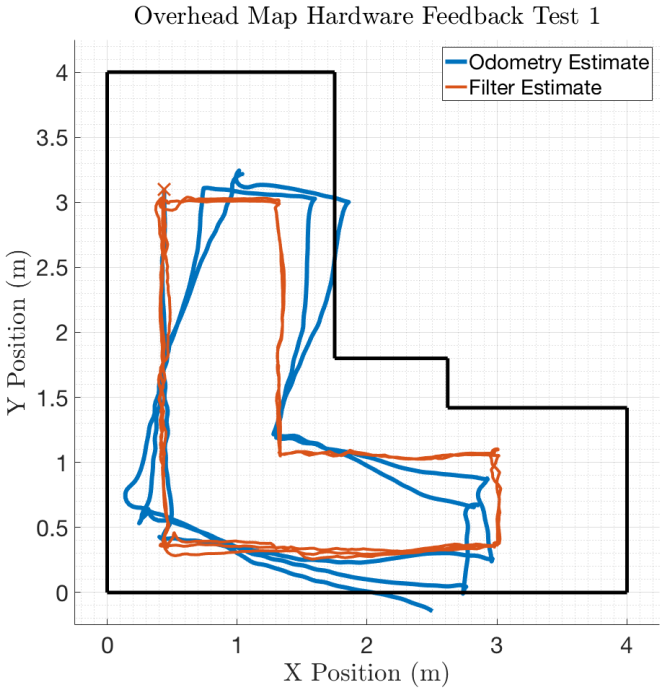



Fig. 14: Hardware particle filter test with a known start. The filter estimate worsens with time because the odometry estimation is used as the state estimate in the point tracker.

For the second feedback experiment, the robot was placed at the same start position with an angle offset of approximately 30 degrees. Despite the offset, the particle filter followed the path in the map. The accumulating odometry error is propagated until it leaves the map. The result is illustrated in Figure 15.

## VII. Conclusion

In simulation, the particle filter has been shown to accurately estimate the state of the robot with an RMSE of $0.0315\text{m}$. Since we lack a ground truth state of the robot in hardware, we cannot quantify the accuracy of the particle filter.

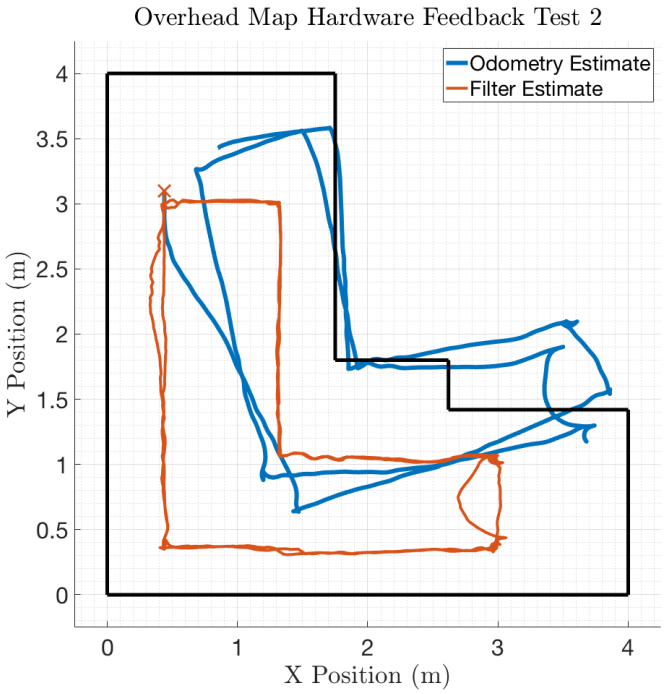



Fig. 15: Hardware particle filter test with a known start. Notice that the filter estimate worsens with time, this is because the odometry estimation is assumed to be the true state.

## References


[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: The MIT Press, 2006.

[2] P. Abbeel, "Beam sensor models," 2011, cs-287 university lecture.

[3] D. Crisan and O. Obanubi, "Particle filters with random resampling times," *Stochastic Processes and their Applications*, vol. 122, no. 4, pp. 1332 – 1368, 2012.

[4] P. Abbeel, "Particle filters," university lecture.

[5] A. Hoover, "Lecture notes: Particle filter," ece854 university lecture notes.