Lab 5 Report: Localization

Team 14

Nick Bonaker
Juliana Chew
Alex Koenig
Kai Maier
John Paris

RSS

April 17, 2021

Contents

1 Introduction (Alex)

Previously, our team enabled a TESSE-simulated car to navigate using visible landmarks including walls, lanes, and cones. In this framework, the car had no knowledge of its position within the world, but rather located the relative positions of objects of interest and followed specified control laws to navigate based on these landmarks. To enable more versatile navigation capabilities – for example, the ability to drive from any arbitrary origin to any destination – the car must first derive its absolute position within the world using a localization algorithm, and then construct a valid route via a motion planning algorithm.

We implement a Monte Carlo localization algorithm to determine the car's location within a known world map, which is the first step towards broader autonomous navigation abilities. Monte Carlo localization is an iterative particle filter process designed to be robust against measurement noise. The key method of Monte Carlo Localization is to randomly sample the distribution of possible car poses (represented by particles) and then update the particle distribution via Bayesian inference. The prior is the particle distribution from the previous time step recomputed based on proprioceptive odometry measurements, the evidence is the car's exteroceptive LiDAR measurements, and the posterior is the new particle distribution. For a more mechanical rather than statistical intuition of this process: at every new time step, the particles are first moved corresponding to the vehicle odometry via a motion model, and then are resampled based on an assessed likelihood of their accuracy by comparing vehicle sensor measurements to the known world map, as per a LiDAR sensor model.

The objective of this lab is to first implement and refine the particle filter (comprised of the motion and sensor model implementation) to enable localization in a simpler 2D environment, and then to adapt the solution to work in the 3D TESSE environment, where it is to be further refined to achieve low localization error and a high rate of convergence. Beginning with an implementation in a simpler environment serves as a stepping stone so that we can more efficiently test whether the key concepts in our implementation are functional before increasing simulation difficulty.

2 Technical Approach

The technical approach is split into several parts: (1) implementation of the motion model, which involves computing the geometrical updates of each particle pose based on measured vehicle odometry at each time step, (2) implementation of the sensor model, which involves computing LiDAR observation probability by comparing the observation to ground truth, and (3) combining each model into an overall particle filter which was implemented in ROS. In each section we test and iterate on the model parameters to enhance the accuracy, robustness to noise, and rate of convergence of the localization algorithm.

2.1 Motion Model (Kai)

Inputs The motion model receives a set of two different inputs: raw odometry particles and an odometry transformation matrix. The raw odometry particles are input as an array containing N poses. Each of these poses are defined by a 3x1 array:

$$X = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} p \\ \theta \end{bmatrix} \tag{1}$$

For each of these poses, we take the pose orientation θ and generate the rotation matrix for the particle which has the form:

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{2}$$

The odometry transformation matrix is similarly input as a 3x1 matrix arranged the same as above. Utilizing these inputs, we generate a new array of N poses which represent the projected positions of the inputted particles. We achieve this through the implementation of a rigid body transformation algorithm.

Rigid Body Transformation The motion model takes in raw odometry particles, including the noise, and transforms it into a set of projected particles.

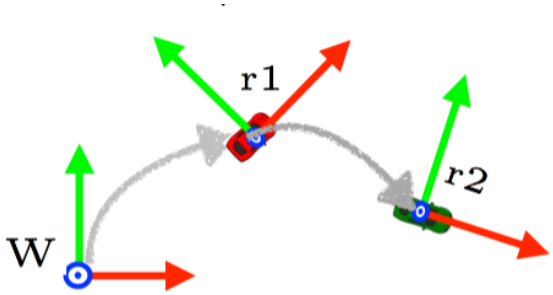

Figure 1: Example set-up showing how rigid body transformation works. In this diagram, W coordinate system represents the 'world' coordinate origin from which the poses for r1 and r2 can be defined. r1 represents the coordinate system of the current pose, and r2 represents the projected pose.

To project the raw odometry particles we preform a rigid body transformation defined by the equation:

$$p_{r2}^{W} = R_{r1}^{W} p_{r2}^{r1} + p_{r1}^{W} \tag{3}$$

$$\theta_{r2}^{W} = \theta_{r2}^{r1} + \theta_{r1}^{W} \tag{4}$$

In this equation p_{r2}^{W} and θ_{r2}^{W} define the pose of the projected particle, p_{r1}^{W} and θ_{r1}^{W} define the pose input by the raw odometry, and p_{r2}^{r1} and θ_{r2}^{r1} come from the transformation matrix which is also provided by odometry.

After undergoing rigid-body transformation, the projected odometry points are then input into the sensor model, which mitigates the noise from the odometry data and attaches a probability to each point based off the ground truth.

Noise for 2D In order to properly synthesize realistic odometry data for testing in RVIZ, noise is added to the motion model to add uncertainty for the sensor model and prevent it from converging towards a local (rather than global) maximum.

2.2 Sensor Model (Juliana)

The sensor model takes the Motion Model's particles as input and determines the value $P(z_k|x_k, m)$, the likelihood of the particle with measurement z_k given the ground truth observations x_k and map m. These likelihoods are fed into the Particle Filter so that particles with lower likelihoods are pruned.

The Look-up Table An efficient run time is essential for real-time localization. As the sensor model involves a great number of calculations for all particle rays, we reduce computational costs by discretizing distance and precomputing a likelihood table. This table considers all possible combinations of ground truth d and measured distances $z_k^{(i)}$ up to a specified maximum range z_{max}, as formatted in Figure 2. By doing so, the sensor model need only "look up" values for each particle ray.

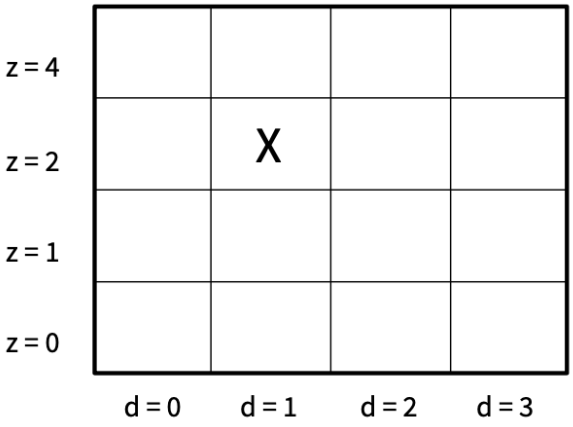

Figure 2: The format of the look-up table of probabilities, where (in this case) the maximum range $z_{max} = 3$. The "X" is $P(z_k^{(i)} = 2|x_k^{(i)} = 1, m)$. Each column is normalized to sum to 1.

This look-up table is discretized into pixels — not meters. To scale measurements and ground truth distances to pixels, we use the following conversion:

$$d_{px} = \frac{d_m}{\zeta\lambda} \tag{5}$$

Where d_{px} and d_m is the given distance in pixels and meters, respectively. The constants ζ and λ are the map resolution (in meters per pixel) and the ratio of the LiDAR scale to the map scale. The values ζ and λ are given because they are constants particular to the map and TESSE. In the 2D simulation, $\lambda = 1$ and $\zeta = 0.05$; in TESSE, $\lambda = 5$ and $\zeta = 0.0967$.

The sensor model performs ray tracing to produce the LiDAR measurements that the robot would have if it were oriented at each particle. From there, the sensor model evaluates the mismatch between these measurements and the ground truth observations to determine the particle's likelihood.

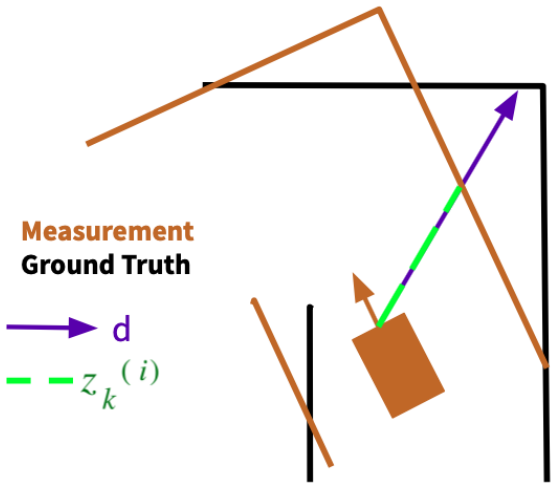

Figure 3: An example of a proposed particle's measurement as compared to ground truth observations. For each ray, the sensor model compares the ground truth distance d to the measurement's ray $z_k^{(i)}$ to calculate a probability for each ray (denoted as $P(z_k^{(i)}|x_k^{(i)} = d, m)$). There is a high mismatch in the figure above, so the particle is assigned a low overall likelihood.

Calculating Likelihoods Each probability $P(z_k^{(i)}|x_k^{(i)} = d, m)$ is defined as the weighted sum as follows:

$$\begin{aligned} P(z_k^{(i)}|x_k^{(i)}, m) = \alpha_{hit} P_{hit}(z_k^{(i)}|x_k^{(i)}, m) \\ + \alpha_{short} P_{short}(z_k^{(i)}|x_k^{(i)}, m) \\ + \alpha_{max} P_{max}(z_k^{(i)}|x_k^{(i)}, m) \\ + \alpha_{rand} P_{rand}(z_k^{(i)}|x_k^{(i)}, m) \end{aligned} \tag{6}$$

Such that

$$\alpha_{hit} + \alpha_{short} + \alpha_{max} + \alpha_{rand} = 1 \tag{7}$$

Where

- $P_{hit}(z_k^{(i)}|x_k^{(i)} = d, m)$ is the probability of detecting a known map obstacle.
- $P_{short}(z_k^{(i)}|x_k^{(i)} = d, m)$ is the probability of a short measurement, often caused by internal reflections, reflections with the robot, or unexpected obstacles.

- $P_{max}(z_k^{(i)}|x_k^{(i)} = d, m)$ is the probability of a large (or missed) measurement. These instances often occur due to unexpected reflecting properties, and the LiDAR instrument does not receive feedback.
- $P_{rand}(z_k^{(i)}|x_k^{(i)} = d, m)$ is the probability of a random measurement.
- α_X is the weight for the probability $P_X(z_k^{(i)}|x_k^{(i)} = d, m)$.

All α_X values are tunable parameters. However, we found that differing values of α did not strongly affect performance. Therefore, we opt to keep all α weights to the initial given values stated below.

$$\alpha_{hit} = 0.74 \tag{8}$$
$$\alpha_{short} = 0.07 \tag{9}$$
$$\alpha_{max} = 0.07 \tag{10}$$
$$\alpha_{rand} = 0.12 \tag{11}$$

The probability P_{hit} is defined as a Gaussian centered on the ground truth distance d, where $\sigma = 0.5$m. With this design, greater agreement between the measurement $z_k^{(i)}$ and d lead to greater likelihoods:

$$P_{hit}(z_k^{(i)}|x_k^{(i)} = d, m) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(z_k^{(i)} - d)^2}{2\sigma^2} & \text{if } 0 \leq z_k^{(i)} \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Unlike P_{hit}, P_{short} is defined as a linear equation, where higher probabilities are located closer to $z_k^{(i)} = 0$ (i.e. closer to the car).

$$P_{short}(z_k^{(i)}|x_k^{(i)} = d, m) = \begin{cases} \frac{2}{d}(1 - \frac{z_k^{(i)}}{d}) & \text{if } 0 \leq z_k^{(i)} \leq d \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

The probability P_{max} represents the probability of a long or missed measurement. Given a set maximum range z_{max}, the only time this type of measurement occurs is when $z_k^{(i)} = z_{max}$. In essence, this creates a δ function centered at z_{max}. We therefore define P_{max} as follows:

$$P_{max}(z_k^{(i)}|x_k^{(i)} = d, m) = \begin{cases} 1 & \text{if } z_k^{(i)} = z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

We also model random measurements P_{rand} as a uniform distribution over all distances in the table.

$$P_{rand}(z_k^{(i)}|x_k^{(i)} = d, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_k^{(i)} \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

After finding the likelihood $P(z_k^{(i)}|x_k^{(i)} = d, m)$ for each ray, we calculate the likelihood of the particle as follows:

$$P(z_k|x_k, m) = \prod_{i=1}^{n} P(z_k^{(i)}|x_k^{(i)}, m) \tag{16}$$

From the equation above, we consider the entirety of the particle's scan. The probability for each particle is then handed to the Particle Filter for resampling.

Flattening the Distribution After precomputing the look-up table, we perform flattening on the probability distributions.

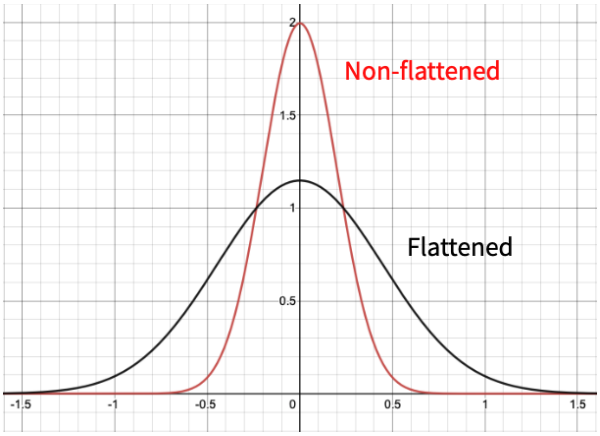

Figure 4: A visualization of flattening. Peaks are smoothed over, and likelihoods are more evenly distributed.

As the particle filter resamples particles by their assigned likelihoods, a more even likelihood distribution allows particles with lower probabilities to not be pruned as prematurely and encourages exploration with more particles in subsequent time steps.

Flattening is defined as follows:

$$P_f(z_k|x_k, m) = P(z_k|x_k = d, m)^{\frac{1}{\beta}} \tag{17}$$

Where β is the flattening parameter, $P(z_k|x_k, m)$ is a given probability in the original table, and $P_f(z_k|x_k, m)$ is the new flattened probability.

We tune the particle filter with varying β values, choosing $\beta = 2.5$. For more information on our tuning strategies and results, refer to section 3.2.1.

Our final probability distribution after tuning can be represented in the 3D plot below:

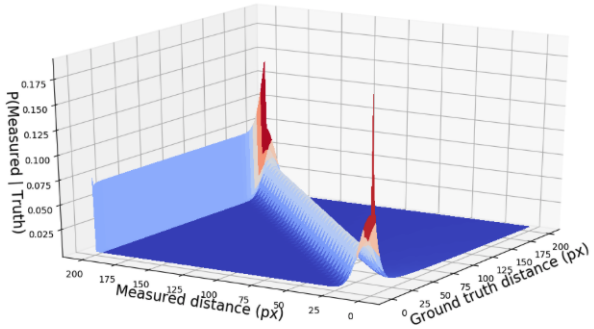

Figure 5: The 3-dimensional representation of the sensor model's look-up table for all combinations of ground truth distance d and measured ray distance $z_k^{(i)}$ in pixels. Gaussian-like peaks are centered on d, and barriers at $z_k^{(i)} = 200$ model missed measurement probabilities.

2.3 Particle Filter (Alex)

The particle filter combines the motion model and sensor model into a single process that is repeated at each time step in the simulator. The essence of the process is Bayesian inference which updates the prior pose belief x_k based on new odometry measurements u_k and sensor measurements z_k. Mathematically, the Bayesian inference procedure is as follows:
The prediction step:

$$p(x_k|u_{1:k}, z_{1:k-1}) = \int p(x_k|x_{k-1}, u_k)p(x_{k-1}|u_{1:k-1}, z_{1:k-1})dx_{k-1} \tag{18}$$

The update step, where α is a constant:

$$p(x_k|u_{1:k}, z_{1:k}) = \alpha p(z_k|x_k)p(x_k|u_{1:k}, z_{1:k-1}) \tag{19}$$

Computing these values analytically is intractable in general. To model this process in a computable manner, we use Monte Carlo random sampling: we take random samples (particles) from the prior belief, and compute the updates for each particle individually so as to get an approximate representation of the posterior distribution. Figure 6 outlines the steps in this process:

- First, the initial belief—comprised of particles sampled from a probability distribution representing our knowledge of the vehicle's pose—is updated with odometry via the motion model. This step contributes noise because odometry measurements are inherently noisy, and therefore it degrades our

knowledge of the vehicle's pose. In this step we even add more noise on top of the measurement noise so as to prevent the belief distribution from being stuck at a local maximum. Say, for example, the vehicle believes itself to be on one street but it is actually on a different street. The believed street is a local maximum in the sensor model likelihood, and the way to escape it is by adding noise such that some particles end up on the correct street (the global maximum), where the distribution will eventually converge.

- Second, the sensor measurements (represented at each particle pose) are compared against the world map via the sensor model to determine the likelihood the particle pose is the vehicle's pose. Although sensor measurements are inherently noisy as well, this step tends to reduce error because it provides evidence sampled directly from the ground truth (the LiDAR measurements are from the car's exact location, not the individual particle locations).

- Finally, the particles are resampled from this new posterior probability distribution. This step is necessary to take into account the updated particle likelihoods.

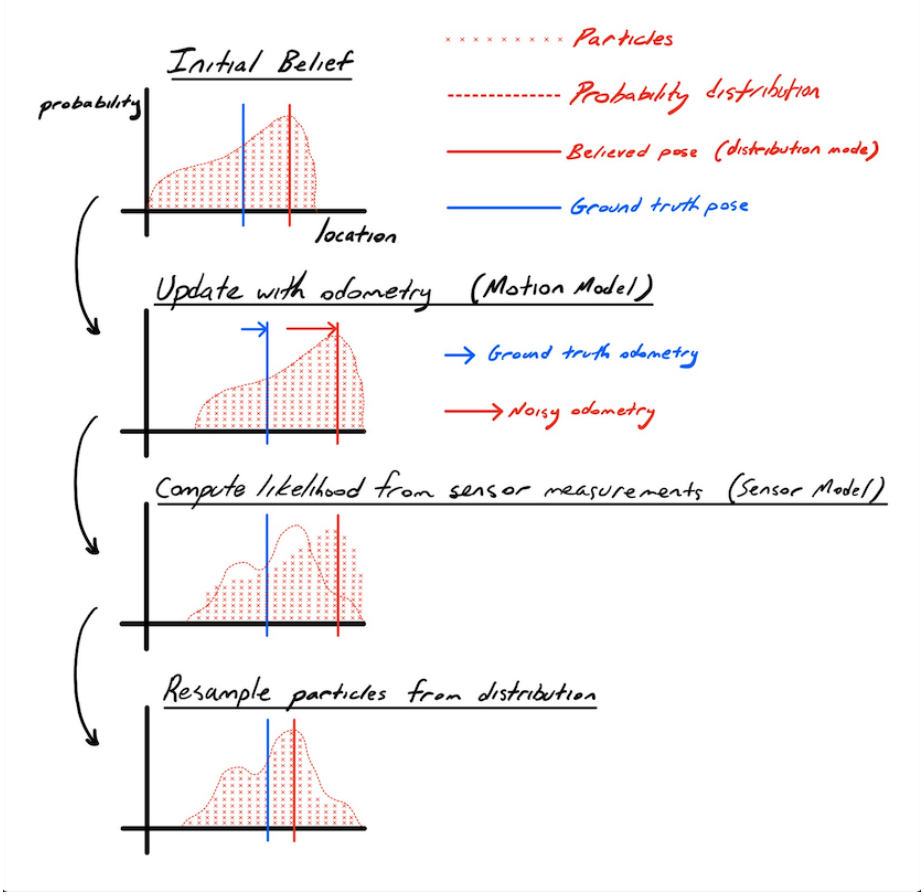

Figure 6: The Particle Filter represented as a Bayesian inference process. This process is iterative, so the last step serves as the starting point for the process in the next time step. The odometry step tends to increase error, whereas the sensing and resampling steps tend to reduce it.

3 Experimental Evaluation

3.1 Simulation Method (Nick)

We develop a simulation framework to evaluate the performance of our particle filter in a consistent manner. This framework allows us to tune various parameters of the particle filter and increase its performance.

To build this framework, we first recorded rosbags of the racecar simulator as we drove the car around the map. These rosbags provided consistent odometry and LiDAR data which we fed as input to our particle filter. We then wrote a suite of bash scripts to manage the setup and execution of our particle filter with the bagged data. These scripts accept a parameter file that passes parameter values (flatten or noise coefficient, for instance) to the particle filter for testing. After running the particle filter on the bagged data, the suite exports both the particle filter's pose predictions and the ground-truth poses to a csv file. We then analyzed this csv file to construct the plots used for tuning in this section.

3.2 Tuning the Particle Filter (Nick)

We use our simulation framework to tune the major parameters in both the sensor model and the motion model of the particle filter. We investigate the effect of varying the flattening coefficient of the sensor model in section 3.2.1 and the effect of varying the internal noise coefficient of the motion model in section 3.2.2.

3.2.1 Sensor Model: Flattening Coefficient (Juliana)

Tuning the flattening parameter β is important in encouraging particle exploration in our localization algorithm. A higher β "squashes" abrupt particle peaks more than lower β. This "squashing" effect distributes likelihoods more evenly across particles and allows more particles to survive in later time steps. For more information, refer to section 2.2.

The figure below shows our particle filter's performance with different β values.

Figure 7: The particle filter performance in the 2D simulation with varying flatten exponent denominators. Performance is measured as root-mean-square error in position and orientation.

Indeed, the figure above demonstrates that flattening increases the particle filter's performance to a certain extent. When there is no flattening ($\beta = 1$), the particle filter's error is higher as the particle filter eliminates particles prematurely. Having too high of a β increases the filter's error as well, because the particle filter eliminates too few particles, including those that are far off from the ground truth pose.

From the figure above, the lowest error in position and orientation occurs when $\beta \in [2.25, 2.75]$. However, the orientation and position error varies in opposite directions within this β range. To compromise between the two, we chose $\beta = 2.5$. This design choice yields a root-mean-square position error of approx-

imately 3.6 degrees and orientation error of about 0.21 meters, with a range of about 2.2 degrees and 0.45 meters. With a high accuracy and relatively high precision, our particle filter appears to perform quite well with $\beta = 2.5$.

3.2.2 Motion Model: Internal Noise Coefficient (Nick)

The particle filter relies on randomness to explore the state space of possible poses. By injecting artificial noise into the motion model, we can control how much the particles spread out. The internal noise coefficient controls the standard deviation of the Gaussian noise added into the motion model. Too little noise, and the particles will remain clumped together and can miss potentially better pose estimates. Too much noise, and the particles will spread out and not converge to an agreement. As evident in figure 8, our simulations showed a noise level in the range of [0.04, 0.09] meters provided the lowest MSE pose estimate. We chose a noise coefficient value of 0.05 meters for our motion model.

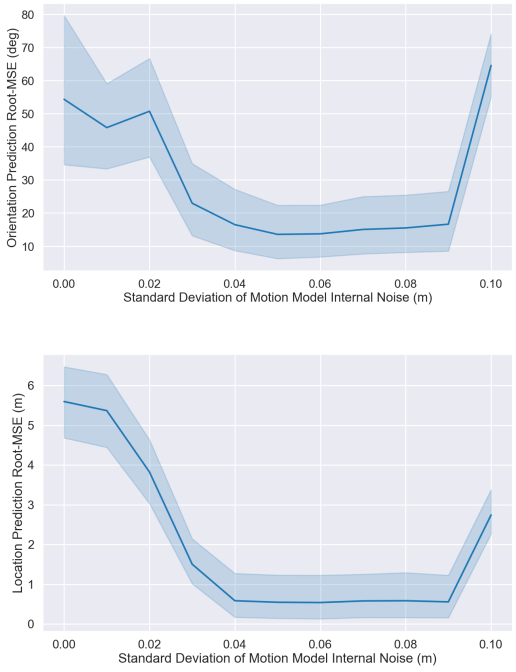

Figure 8: Effects of varying the internal noise coefficient of the motion model on the particle filter performance. The noise coefficient controls how much the particles spread out. If the noise is too low, the particles will remain clumped together and potentially miss better pose estimates. If the noise is too high, the particles will spread out and may not converge. Our simulations show a noise level of around 0.05 m provided the lowest MSE pose estimate.

3.3 2D Evaluation (Nick)

Using these tuned flattening and internal noise coefficients in our particle filter, we evaluated its performance with varying levels of odometry noise. In the real world, the car will not know its exact velocity and pose at a given time – we simulated this error by adding Gaussian noise of increasing variance to the car's odometry data. As evident in figure 9, the particle filter's performance is robust to noise with up to 2 whole meters of standard deviation. The location predictions achieve a near constant root-mean-square-error of 0.22 meters (only 8 inches), and the orientation predictions a root MSE of 11 degrees.

With this level of precision and resiliency to noise in our 2D particle filter, our car should have no problems with performing localization even for high-demand tasks such as parallel parking in the crowded streets of Boston.

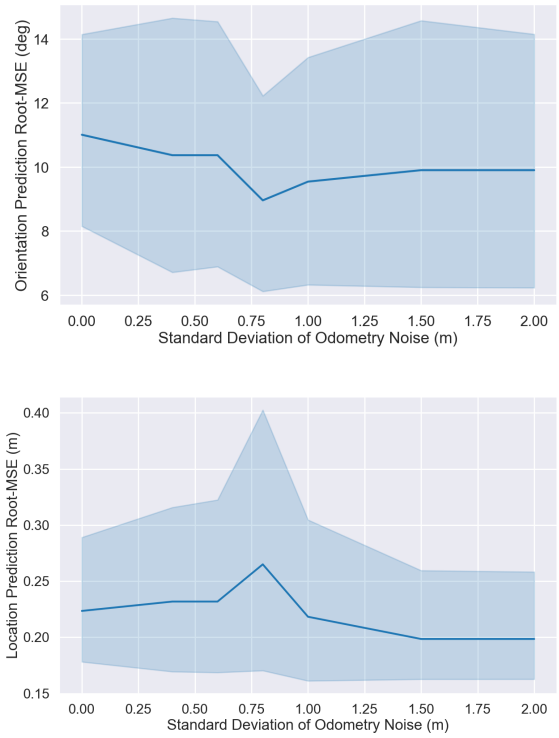

Figure 9: Performance of the tuned particle filter with varying degrees of odometry noise. Our particle filter is quite accurate and extremely resilient to noise. It achieves a near constant location root-MSE of 0.22 meters (only 8 inches) and orientation root-MSE of 11 degrees.

3.4 3D (TESSE) Evaluation (Alex)

When implementing our code in the TESSE environment, we encountered numerous technical bugs which slowed down the testing and improvement of our code. Nevertheless, we were at least able to test our localization algorithm in TESSE. Since we ran our tests within the staff-provided virtual desktop interface (VDI), we drove the car by publishing drive messages to drive the car in straight lines starting at a variety of locations within the map, and simultaneously ran the localization algorithm while bagging data from the run. The plots below demonstrate the root-mean-square error of both the position estimate and orientation estimate of the car from a particular representative run.

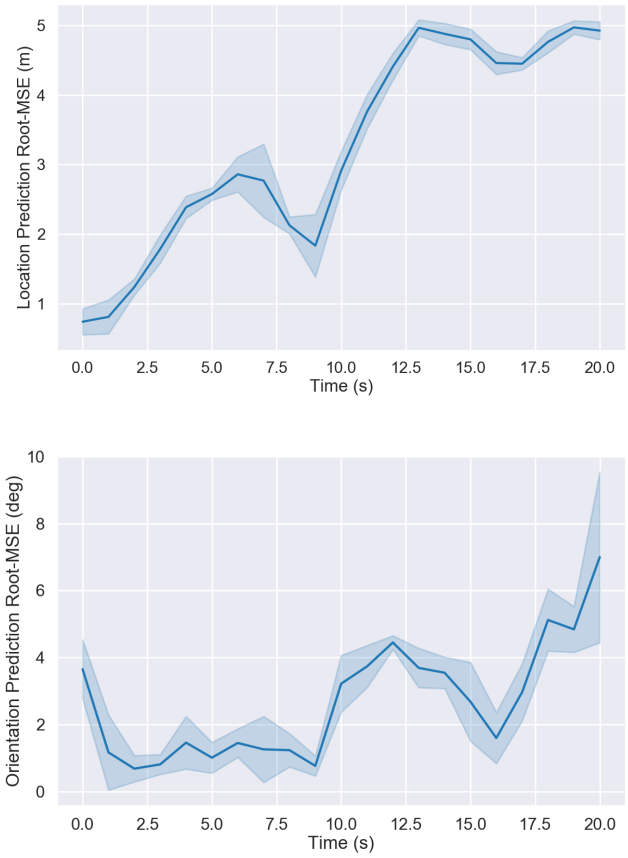

Figure 10: MCL-estimated pose error during a TESSE run.

In general, the orientation prediction is strongly correlated with ground truth, never exceeding more than 10 degrees error, and on average obtaining a 3.7 de-

gree error. This result is highly acceptable. The location prediction is not quite as accurate—it starts out within 1 meter of truth but ends up at roughly 5 meters from it, and has an average error of 2.7 meters—and concerningly it appears to diverge from the actual car location rather than converge upon it (which is the anticipated behavior). We believe that the reason for this divergence is likely due to some sort of as-of-yet undiscovered transformation bug within the code. Based on our localization algorithm's behavior in the 2D environment, we expect that the localization is actually converging to a certain location, but a faulty TESSE-implementation-related transformation pushes that prediction roughly 5 meters away.

Assuming our issue is solely caused by an undetermined transformation problem, it takes roughly 10-15 seconds for the location prediction to converge, and when it converges, the average variance is 0.25 meters. Some credence to this speculation is derived from the fact that in the 2D environment, the average post-convergence position error was 0.22 meters; if anything, the 3D environment is somewhat more complex and noisy, so we would expect a marginally greater error. Further credence to this speculation is given by the RVIZ screenshots in Figure 11. These screenshots show that although the pose error is consistently high (roughly 5 meters), the estimate is still highly constrained and only has high error because the estimated position is consistently roughly 5 meters in front of the car itself, no matter the orientation.

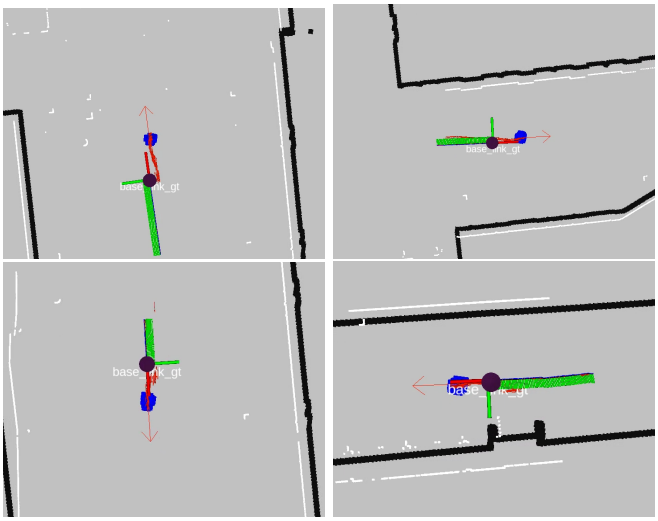

Figure 11: MCL-estimated pose during a TESSE run. Blue markers are individual particles, the red arrow is the estimated pose, and base_link_gt is the car location (the red axis is the forward direction). In all scenarios, the pose is consistently 5 meters in front of the car, but is otherwise accurate.

4 Conclusion (John)

This lab progressively walks through the creation of a Monte Carlo localization algorithm. Implementing a Monte Carlo localization scheme will allow us to more easily implement complex path planning algorithm in the future. By keeping track of where we are, we allow ourselves to create planning algorithms for obstacles we cannot see in the current time step, given map data.

In order to design this Monte Carlo localization system, we learn both how to create a motion model that used odometry to predict where points in previous time steps should be in the future as well as a sensor model that indicated the probability of LiDAR sensor data showing up in particular locations given a ground truth location. The combination of the two models allows us to determine the probability of the car's ground truth location given a series of LiDAR data.

Our implementation of this Monte Carlo localization in a particle filter in 2D has been measured to be quite accurate. However while we were able to implement our code into TESSE, our 3D implementation of the particle filter could use improvement. This can be seen by our root-mean-square error of ± 2.7 meters. With this error, the car can localize itself on a road (requires ~ 3.7 meters), but it cannot localize itself within a lane (requires ~ 1.85 meters, or half a standard lane width). We can improve the 3D system by ensuring that the particle filter uses up-to-date LiDAR data and up-to-date position transforms to relate the noisy TESSE LiDAR data to real-time map orientations.

5 Lessons Learned

Alex This lab strongly reinforced that I should never underestimate the integration step as part of the overall technical work. Combining two perfectly working components does not guarantee a perfect whole, since a working solution also relies on the successful interaction of those components; similarly, just because a solution works in one environment does not entail that porting it to another environment will be straightforward and trouble-free. Regarding CI lessons, I felt our briefing was much more successful this lab because we placed a greater emphasis on the holistic approach, context, and motivation for our work rather than highly specific details about the particulars of our implementation.

Juliana I learned (quite quickly) that the integration step is one of the hardest and most taxing parts of a collaborative project. We underestimated the time it would take to run the working particle filter in TESSE, and many of us ran into unique TESSE-related bugs that prevented us from integrating until the last minute. If anything, it would have worked better if we budgeted more time for this phase and also if we had asked our TAs for help earlier. Words cannot

describe how appreciative I am of the TAs' responsive help in the odd hours of the night!

John The largest lesson I learned is to ask for help on technical errors early. I struggled a lot with a series of Tesse-ROS errors that could have been resolved much sooner if I'd asked for help earlier. From a RSS theory standpoint I felt that I got a very good understanding of the theory behind Monte Carlo Localization and how to implement it.

Nick I primarily focused on the integration evaluation of the 2d particle filter in this lab. I learned that time is the most valuable resource to a team, and that earlier parts should be finished ASAP in case there are errors down the line. Working to finish the 2d particle filter implementation earlier in the week would have allowed more time to evaluate and tune it and the process would have felt less time-crunched.

Kai While I primarily focused on the implementation of the motion model in this lab, the team as a whole learned a great deal about the difficulties included in meshing together the different parts of code. However I also learned through the motion model that there are always more than one correct implementation of the same equation. I struggled with making the matrix form of rigid-body transformation work however when i transitioned to a different implementation of the same algorithm it work perfectly. Additionally I am very excited about the ability to use VDI since it enables me to finally help test the code in TESSE.