Lab #5 Report: Localization

Team 4

Toomas Tennisberg
Frank Gonzalez
Marisa Hoosen
Zoe Wong
Kaitlin Zareno

RSS 6.141

April 1, 2022

1 Introduction

Written By: Kaitlin Zareno

In this lab we explored localization through implementing a Monte Carlo Localization Algorithm. Localization is the process of determining a robot's orientation and position in a known environment. This is a helpful component because autonomous localization is required for autonomous navigation. As we begin to build the foundation of these different skills, we can begin to see how the skills will piece together to allow us to reach our final objective: autonomously race and maneuver our car. In the past few labs we learned how to autonomously follow a wall and a "line" created by cones, and now with localization, we are able to properly understand the relative pose of the car in space and recognize the car's placement relative to important landmarks (such as a cone it would like to park in front of).

In the following sections we will discuss our implementation of the Monte Carlo Localization algorithm, and then how we evaluated it, both in simulation and in reality.

2 Technical Approach

2.1 Introduction

Written By: Kaitlin Zareno

In order to implement Monte Carlo Localization (MCL) we needed a motion model, sensor model, and particle filter. The motion model's objective is to deterministically infer the robot's position in space from its odometry measurements and update the robot's pose. The purpose of the sensor model is to define the likelihood of recording a given sensor reading from a hypothesis position, which is important in validating the actual pose of the robot in space. Lastly, we combine the motion model and sensor model in the particle filter to carry out MCL and over time allow the particles that are being used to infer the car's position in space to converge to the car's actual location.

2.2 Motion Model

Written By: Frank Gonzalez

The motion model is in charge of taking a set of particles and updating their poses given a local odometry input. The updating procedure is governed by the following equation:

$$\begin{bmatrix}\Delta x & \Delta y & \Delta\theta\end{bmatrix}\begin{bmatrix}\begin{bmatrix}\cos\theta_i & -\sin\theta_i & 0\\ \sin\theta_i & \cos\theta_i & 0\\ 0 & 0 & 1\end{bmatrix}\\ \ldots\end{bmatrix} + \begin{bmatrix}x_1 & y_1 & \theta_1\\ x_2 & y_2 & \theta_2\\ & \ldots & \end{bmatrix} \tag{1}$$

Here, we take the local odometry, the matrix on the left, and apply it to a series of rotation matrices. Each rotation is defined according to the corresponding particle's theta component. Then, this transformed odometry is summed into particle's original pose. In addition to this update, a noise matrix is also constructed in the manner shown below. This matrix can optionally be applied at the end of the computation to simulate noisy odometry, which later on will be key for the particle filter's performance:

$$\begin{bmatrix}| & | & |\\ \mathcal{N}(0, 0.01) & \mathcal{N}(0, 0.01) & \mathcal{N}(0, 0.002)\\ | & | & |\end{bmatrix} \tag{2}$$

2.3 Sensor Model

Written By: Zoe Wong

The sensor model is responsible for computing how likely a given pose is the actual pose of the robot. It determines this likelihood based on how probable it is to record the given LIDAR measurements at this pose in the given static map.

2.3.1 Calculations of Likelihood of Observing Measurement at Hypothesized Pose

The probability of observing a given scan at a hypothesized pose is defined as the product of the likelihoods of measuring each of the n LIDAR measurements.

$$p(z_k|x_k, m) = p(z_k^{(1)}, \cdots, z_k^{(n)}|x_k, m) = \prod_{i=1}^{n} p(z_k^{(i)}|x_k, m) \tag{3}$$

In this equation, $p(z_k|x_k, m)$ represents the probability of recording the measured LIDAR scan z_k at hypothesized pose x_k in the given map m at time k. Furthermore, $p(z_k^{(i)}|x_k, m)$ indicates the likelihood of reading the i^{th} range measurement of the scan at this guess pose.

To compute for $p(z_k^{(i)}|x_k, m)$, we considered the following 4 main cases:

1. *The LIDAR correctly detects the closest known object in the map.*

   Due to the limitations of the LIDAR and possible non-ideal circumstances in the environment, there could be some error in the measurements, meaning the measured ranges may not exactly match the actual distances. Thus, we represented the likelihood of correctly detecting the closest known object as a Gaussian distribution centered around the actual distance d between the hypothesis pose and the known obstacle. With this representation, the probability is set as the maximum when the measured range $z_k^{(i)}$ is equal to the ground truth distance d, and decreases exponentially as the error, or the difference between the measured and actual distances, increases.

$$p_{hit}(z_k^{(i)}|x_k, m) = \begin{cases}\eta\dfrac{1}{\sqrt{2\pi\sigma^2}}e^{-\dfrac{(z_k^{(i)}-d)^2}{2\sigma^2}} & \text{if } 0 \le z_k^{(i)} \le z_{max}\\ 0 & \text{otherwise}\end{cases} \tag{4}$$

The η represents a normalization factor, such that the sum of $p_{hit}(z_k^{(i)}|x_k, m)$ for all values of $z_k^{(i)}$ and a particular actual distance d equals 1.

2. *We get a small measurement due to the LIDAR's internal reflections or due to an unexpected obstacle in the way of the path of the LIDAR beam to the closest known object.*

   If unknown objects are distributed uniformly along the path between the robot and the known obstacle, the LIDAR is more likely to detect ones that are closer. Therefore, we represented the chances of this case happening as a downward sloping line as the LIDAR beam gets further from the robot, or as $z_k^{(i)}$ increases.

$$p_{short}(z_k^{(i)}|x_k, m) = \frac{2}{d}\begin{cases} 1 - \frac{z_k^{(i)}}{d} & \text{if } 0 \leq z_k^{(i)} \leq d \text{ and } d \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

   Similar to the η in the equation for the $p_{hit}(z_k^{(i)}|x_k, m)$, the $\frac{2}{d}$ is a normalization factor.

   By definition of integrals, the sum of $1 - \frac{z_k^{(i)}}{d}$ for all values of $z_k^{(i)}$ where $0 \leq z_k^{(i)} \leq d$ is equal to its integral. Since this function is a line, its integral is also the triangular area between the line and x-axis. When $z_k^{(i)} = 0, 1 - \frac{z_k^{(i)}}{d} = 1$. When $z_k^{(i)} = d, 1 - \frac{z_k^{(i)}}{d} = 0$. Thus, $\int_0^d 1 - \frac{z_k^{(i)}}{d} = \frac{1}{2}(1-0)(d) = \frac{d}{2}$. Therefore, for the final sum of the probabilities to be equal to 1, the normalization factor must be $\frac{2}{d}$ because $\frac{2}{d}(\frac{d}{2}) = 1$.

3. *We get a very large measurement because the LIDAR beam likely hit an object with strange reflective properties and did not reflect back to the sensor. In this case, the LIDAR assumes the closest object in that direction is farther than the maximum distance it can detect.*

   We represented this case as a "large spike in probability at the maximal range, ... approximated by a uniform distribution close to z_{max} for small ϵ".

$$p_{max}(z_k^{(i)}|x_k, m) = \begin{cases} \frac{1}{\epsilon} & \text{if } z_{max} - \epsilon \leq z_k^{(i)} \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

4. *We get a completely random measurement.*

   For this scenario, we assumed a uniform distribution from 0 to the maximum range measurement z_{max}.

$$p_{rand}(z_k^{(i)}|x_k, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_k^{(i)} \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

After accounting for these 4 main scenarios, we define $p(z_k^{(i)}|x_k, m)$ as below.

$$p(z_k^{(i)}|x_k, m) = \alpha_{hit} \cdot p_{hit}(z_k^{(i)}|x_k, m) + \alpha_{short} \cdot p_{short}(z_k^{(i)}|x_k, m) + \alpha_{max} \cdot p_{max}(z_k^{(i)}|x_k, m) + \alpha_{rand} \cdot p_{rand}(z_k^{(i)}|x_k, m) \tag{8}$$

To ensure that $p(z_k^{(i)}|x_k, m)$ is a probability distribution, we make sure that $\alpha_{hit} + \alpha_{short} + \alpha_{max} + \alpha_{rand} = 1$. As a result, the probability distribution for $p(z_k^{(i)}|x_k, m)$ should look similar to the distribution illustrated in Figure 1.

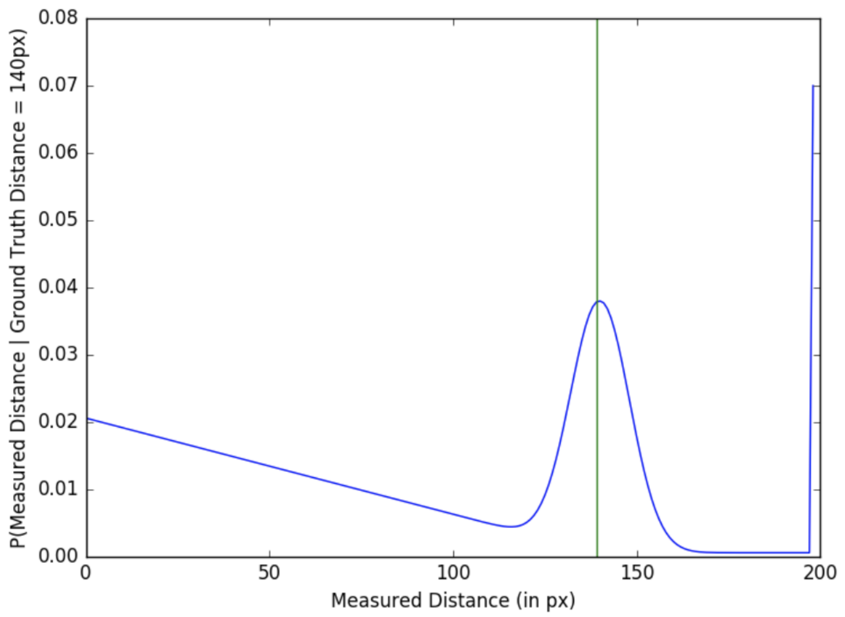


Figure 1: Graph relating the value of a measurement with the probability of the measurement.

2.3.2 Precomputation of Discretized Sensor Table

For our implementation of the sensor model, we precomputed a discretized sensor table. With this, we can reduce the processing time required to compute the particle likeliness to just that of reading from the table given a measured and actual distance. Additionally, with a precomputed table, we can guarantee that the discretized probabilities $p(z_k^{(i)}|d)$ for all values of measured distance $z_k^{(i)}$, given a particular actual distance d, sum to 1 with normalization.

To generate this table, for each actual distance from 0 to 200 pixels inclusive, we computed the likelihood that the measured distance was 0 to 200 pixels inclusive. We substituted the following constant values when calculating $p(z_k^{(i)}|x_k, m)$ using the above equations.

z_{max}	200
σ	8.0
ϵ	1.0

α_{hit}	0.74
α_{short}	0.07
α_{max}	0.07
α_{rand}	0.12

Additionally, when determining p_{hit}, we first computed $\frac{1}{\sqrt{2\pi\sigma^2}}\exp\Big(-\frac{(z_k^{(i)}-d)^2}{2\sigma^2}\Big)$ for each of the 201 values of $z_k^{(i)}$ (0 to 200 inclusive) for a particular actual distance d. We then defined the normalization factor η as the reciprocal of the sum of these values to ensure the probability distribution of p_{hit} sums to 1.

As a result, we created a 3-dimensional probability distribution, similar to that plotted in Figure 2. As you can notice in this plot, there are two main peaks. The first and highest peak is located along the line where the measured distance is equal to $z_{max} = 200$ pixels, the main contributor being p_{max}. The other but lower and less steep peak is located along the diagonal where the measured distance is equal to the actual/ground truth distance, the main contributor being p_{hit}.

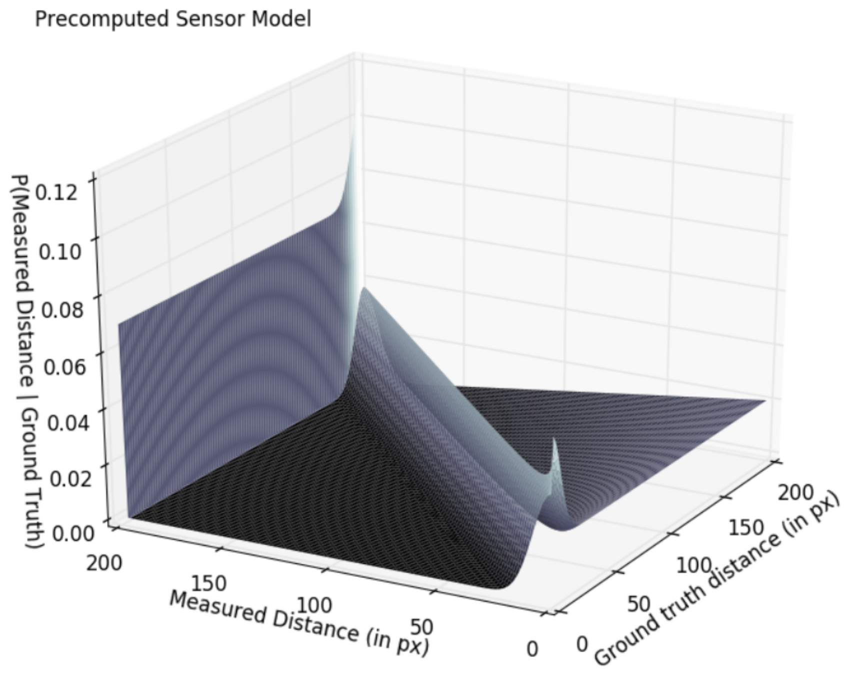


Figure 2: 3D graph relating the actual distance and measured distance to the probability of getting that measurement.

2.3.3 Real-Time Calculations of Likelihood in MCL Algorithm

During each iteration of the particle filter, the sensor model computes the likelihood of a pose hypothesized from the motion model with the real-time LIDAR measurements and the ground truth measurements determined from ray tracing. Since the discretized sensor table has its units in pixels, we need to convert our LIDAR scan and the ray tracing output from meters to pixels. This conversion is dependent on how the given or known map is designed, as well as the map resolution. Using the default maps provided in this lab for simulation and for experimentation in Stata, as well as the default map settings, this conversion was 1-to-1.

After this conversion, for each pair of actual and measured distance recorded at the same angle, we determine $p(z_k^{(i)}|x_k, m)$ by accessing the precomputed sensor table. With $p(z_k^{(i)}|x_k, m)$ for all n range measurement, we then calculated its product to get the finalized likelihood of the hypothesized pose. Additionally, we "squashed" these probabilities by raising them to the power of $\frac{1}{2.2}$. These probabilities will then be used in the particle filter for resampling, as explained below.

2.4 Particle Filter

Written by: Frank Gonzalez

2.4.1 Particle Filter Construction

With both the motion model and sensor model constructed, we can now implement our particle filter, along with the MCL algorithm. The two are outlined below.

The first step in our particle filter's algorithm is to initialize a set of particles at a "guess" location near the car's initial pose. These particles will be used to probabilistically track the car's motion over time, as the particles will move in response to the car's odometry. One key component of the initialization is to ensure

Algorithm 1 Particle Filter

```
1: $X_0$ = initialize_particles
2: for operation_period do
3:     $z_i$ = get_obs()
4:     $u_i$ = get_odom()
5:     $X_i$ = MCL($X_{i-1}, u_i, z_i$)
6:     pose_estimation = calc_estimate($X_i$)
7: end for
```

Algorithm 2 MCL(X_{t-1}, u_t, z_t)

```
1: $X$ = motion_model($X_{t-1}, u_t$)
2: $\omega$ = sensor_model($X_t, z_t$)
3: $X_t$ = sample($X, \omega$)
```

that the particles have a spread around the initial guess. An example initialization is shown in Figure 3:

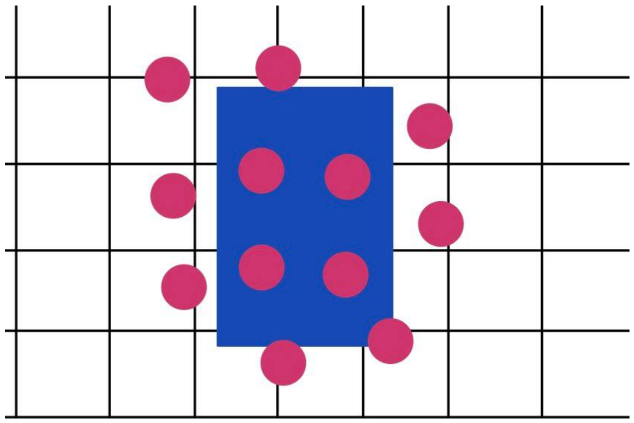


Figure 3: A downsized example of particle initialization.

Then, for each time stamp, the car's odometry along with it's LIDAR sensor data is retrieved. Both are passed into the MCL algorithm, where the updated particle poses along with their probabilities are calculated. Next, the particles are re-sampled with each particle weighted by it's probability to form a new set of updated particles. Over time with this iterative process, the particles will begin to converge around the car's location.

2.4.2 Effect of Noise

One of the particle filter's most fundamental components is its introduction of noise into the system. The addition of noise is important because real life systems tend to contain noise when collecting data, or respond to inputs in a noisy way. Therefore, to ensure that our particle filter is robust to noise, we introduce noise into our simulated particles with motion model. The main benefit to this introduction is the avoidance of particle death. Particle death can be understood with the following image:

If we consider Figure 4, and we assume that odometry input and LIDAR data are noiseless and completely deterministic, we'll notice a huge issue when we resample: some particles are lost while are duplicated. This is problematic because over time, the particles will continue to reduce down until there is only a single particle left. Such a scenario is dangerous because in real life, there is always noise, and with so few particles, bad data could very easily cause our estimations to become inaccurate and even diverge. If we introduce

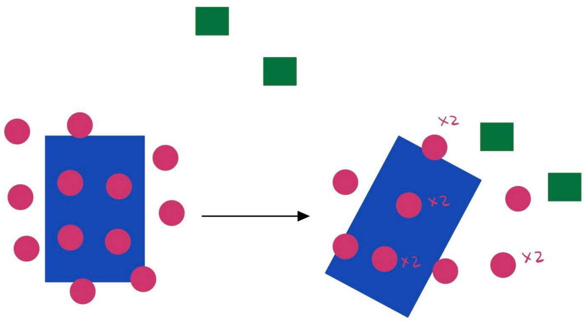


Figure 4: Example of Particle Death.

noise however, if we perturb every particle just a little bit, this prevents the particles from converging into one, and more importantly it allows bad particles to probabilistically undergo a correction of sorts.

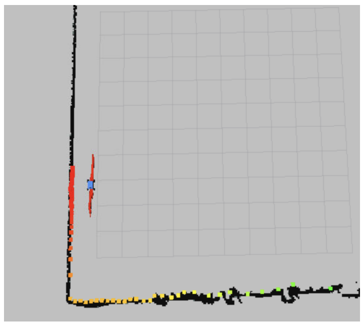
3 Experimental Evaluation

3.1 Particle Filter Simulation Evaluation

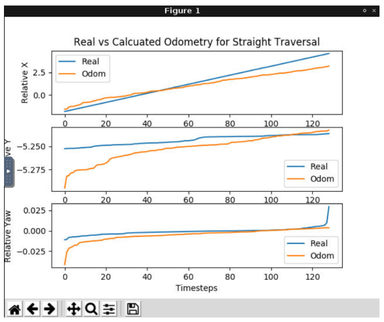
Written By: Kaitlin Zareno

By evaluating our algorithm in simulation, we would be able to gain a better understanding of algorithm and better-tune it for deployment in a real-world environments. Running our algorithm in simulation also allows us to sanity-check our work and make sure the robot is performing as expected before running it in real life, decreasing the risk of unexpected behavior that might cause damage to the robot.

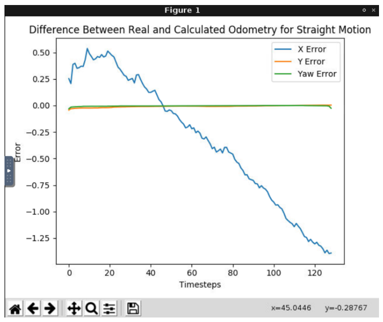
In order to evaluate the particle filter in simulation, we recorded rosbags of the car moving at a velocity of 0.5 m/s in 3 scenarios: straight trajectory, corner traversal, and circular trajectory. These rosbags contained information regarding the real odometry of the car, which was obtained from the tf tree, as well as the calculated odometry data from the particle filter as described above. Given the real and calculated odometry data, we can then begin the evaluation of our algorithm.



(a) Simulation Trajectory



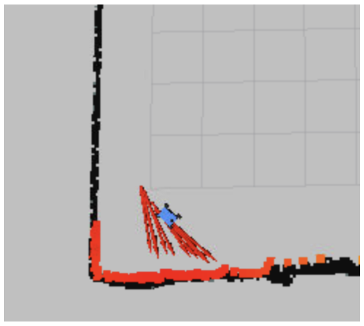
(b) Real vs Calculated Odometry



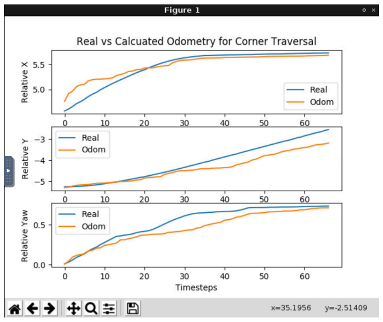
(c) Error Over Time

Figure 5: Straight Trajectory

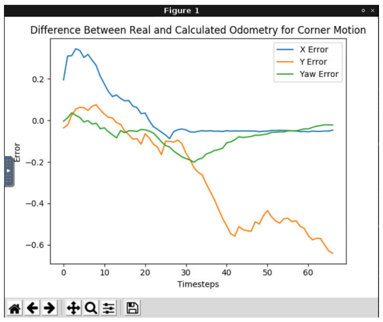
In Figure 5 we can see the simulation trajectory used, real vs calculated odometry data for X,Y, and yaw, and the timestep errors for X,Y, and yaw. In figure 5a we can see that the Y and yaw calculated odometry values begin to converge at around 100 timesteps, and for the X and Y calculated odometry values, we can see that after convergence, divergence begins to occur. In figure 5c, which obtains the absolute error between the real and calculated odometry values, we can see that the Y and Yaw values have minimal error (close to 0) and the X values range in error from +0.5 to -1.25 m. In Figures 6 and 7, we can also see the figures for corner traversal and circular trajectory.



(a) Simulation Trajectory



(b) Real vs Calculated Odometry



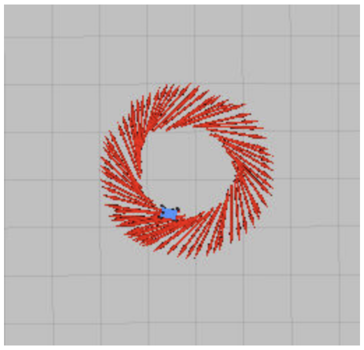
(c) Error Over Time

Figure 6: Corner Traversal

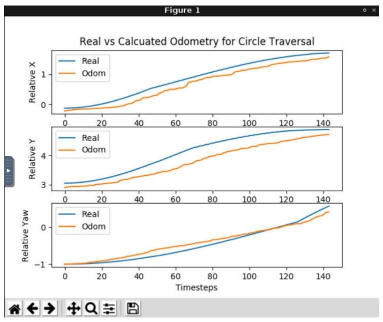
For corner traversal, we can see that the X odometry converged quickly to the real measurements, and the yaw did a similarly good job of converging and maintaining the correct values. On the other hand, we can see that the Y value converges and then diverges to an approximately constant value, depicting a potential steady state error between these values. In the error graph, we can see that the X and yaw errors converge to 0, while the Y error hovers around 0 before dropping off to an error of approximately -0.6 m.
For the circular trajectory, we can see that X and Y calculated odometry values follow the same shape as the real odometry values, but seem to be off by a constant factor, potentially indicating some sort of steady state error with this trajectory. As for the yaw values, we see the values converge at timestep 100 then begin to diverge; that said, the real and calculated odometry plots have a relatively similar slope. In the error graph, we can see that the X, Y, and yaw errors converge to an error value of about -1.5, which may further indicate steady state error as described above. We can also see that the Y plot maintains a negative error, the yaw maintains a relatively positive error, and the X error maintains a negative error centered around about -0.2m.
Lastly, we calculated the average X, Y, and yaw errors at different race car velocities and trajectories.
In the table above we can see that:

3.2 Particle Filter Simulation Improvements

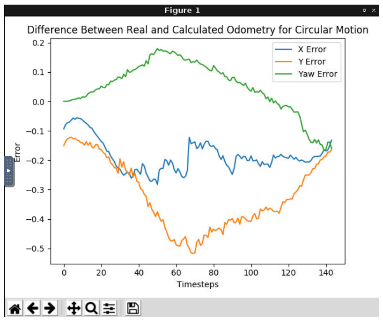
Written By: Kaitlin Zareno



(a) Simulation Trajectory



(b) Real vs Calculated Odometry



(c) Error Over Time

Figure 7: Circle Trajectory

One thing we could do in the future is calculate the convergence of our algorithm. Convergence is important because it would allow us to understand how well our calculated odometry matches the real odometry of the car at steady state, and thus how accurately located our car is within the space over time. Further, fast convergence to the real odometry of the car is important because we want to minimize the time in which the car is less certain about it's location; fast convergence means that the car's calculated odometry would have high confidence in its position in a short amount of time, decreasing the likelihood of motion planning error especially when moving at high speeds.

3.3 Particle Filter Reality Evaluation

Written By: Marisa Hoosen

To test the algorithm on the car in real life, we drove the car in teleop around a corner in the basement of stata while recording rosbags containing LIDAR and odometry measurements. These rosbags were then played in simulation, displaying our predicted locations and orientations on the map, as shown in Figure 7. We were able to successfully navigate a corner and wall containing two open doors and a few indents in the wall.

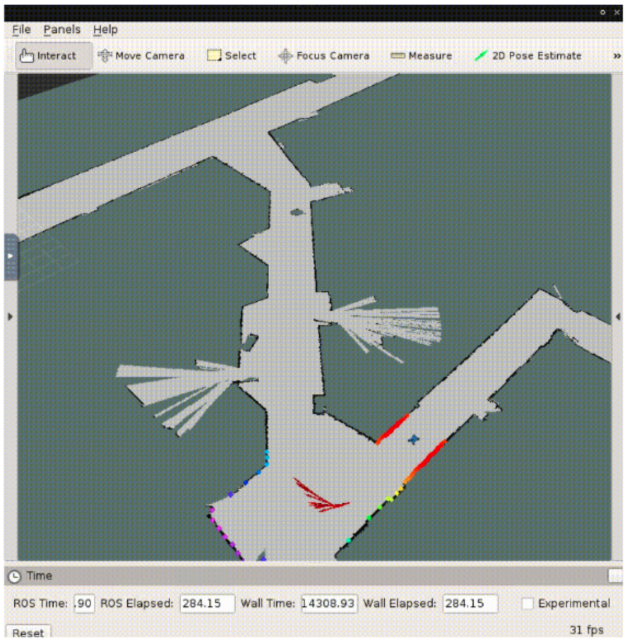


Figure 8: The car's estimated position and orientation in Stata basement as calculated using the particle filter.

3.4 Particle Filter Reality Improvements

Written By: Marisa Hoosen

This real-life testing revealed places for improvement. Our robot had more trouble correctly interpreting tight, 180 degree turns, choosing the correct path from many options, and navigating hallways that turned. Running the robot in a variety of settings with varying landmarks (corners, walls with and without doors and windows, objects in the hallway normally not present, etc.) would potentially find other places of misunderstanding. Running at different speeds would also help ensure that our algorithm is robust and works in many situations.

4 Conclusion

Written By: Marisa Hoosen

Over the course of this lab, we have successfully implemented motion and sensor models, and combined them to create a particle filter.

Our particle filter works in simulation with y and yaw errors of nearly zero along a straight path, yaw error within 0.2m on corner turn, and x, y, and yaw error within 0.5m on a circular trajectory. Our robot is able to successfully localize itself when driven in a random path around the basement of Stata, as depicted by the visualization of its localization replayed in simulation.

The localization could stand to be improved by exploring ways to better manage sharp turns and additional open doorways and paths.

5 Lessons Learned

Presents individually authored self-reflections on technical, communication, and collaboration lessons you have learned in the course of this lab.

5.1 Frank Gonzalez

I really enjoyed seeing an application of probabilistic models to estimate states. I had taken an autonomy and decision making course in a previous semester, but it was very theory based with little room made for application. This lab gave me the opportunity to finally apply some of the knowledge I had obtained.

For communication, I became more effective at expressing my ideas for implementation and keeping concepts clear when presenting them to other individuals.
This lab was definitely the most challenging one so far, as the implementation is relatively complex and solves a much more difficult problem than previous labs. Despite the difficultly, it was exciting to see the progress our localization model would make both in simulation and in reality.

5.2 Marisa Hoosen

This lab, I found it interesting to learn about and implement localization on the racecar. I learned a lot this lab about testing and applying what was tested in simulation to reality, and got practice troubleshooting the differences between them.

Communication wise, I learned about ways to improve our presentation and gotten better at presenting the information and determining the degree of detail to fit the information into the time allotted.

For this lab, we divided up the motion and sensor models into two groups to do to start, but soon after we'd made some progress, we switched to all working on the sensor model and particle filter together after finding that it was more involved and harder to get working than we'd thought. In doing so, we bounced around and all worked on the lab in incremental pieces, advancing the same code more than in the other labs rather than working on separate pieces and joining them together.

5.3 Toomas Tennisberg

I found the refresher on how odometry data needs to be converted to absolute coordinates helpful. I am not that strong with rotations and the numbers always feel like magic to me. I also got a good refresher on probability which ensured that I wouldn't forget my knowledge of 6.008.

On the CI side I learned to communicate with my team better and organize the tasks so that we could get close to finishing on time.

5.4 Zoe Wong

This lab was probably the most difficult among the labs we have worked so far in terms of collaboration. Considering that this lab took place during Spring Break, many of our team members were away on vacation and thus, were, understandably, not able to work on it as much as desired, especially on parts in which we needed to test the physical robot. Additionally, many of us had midterms and assignments due the week right before and after break, making it even more difficult to find time to collaborate together on the project. As a result, we were scrambling to finish the lab and meet deadlines this week. Despite this, I believe we made considerable progress with our lab due to our teammate's commitment. For example, we each made sure to prepare our slides and parts for the lab briefing on our time, given that we had no time to practice until the day of.

5.5 Kaitlin Zareno