Lab 5 Report: Monte Carlo Localization

Team 10

Jeremiah Budiman
Miranda Cai
Tiasa Kim (Editor)
Bill Kuhl
Savva Morozov

MIT 6.141/16.405
Robotics: Science & Systems

April 4, 2022

1 Introduction

Miranda Cai

1.1 Objective

In this lab, we aimed to accurately determine the position and orientation of the robot relative to its environment, a process also known as localization. Localization is an important aspect of robotics and is necessary for problems that involve making future decisions. For example, in Lab 3: Wall Follower, one of the biggest obstacles our team faced was initiating a turn quick enough when approaching a corner at a high velocity. In this lab, we relied on noisy sensor data to notify the car of when to change its steering angle as an immediate response to a change in our local environment. Unfortunately this meant that when moving at high speeds, the racecar was unable to meet the very small response times necessary to send the controls. However, this issue could have easily been solved with localization, since knowing the car's location throughout would have allowed the racecar to determine its position from the corner and thus calculate at what location in the world frame should it start making its turn. Similarly, in Lab 4: Visual Servoing, we were asked to implement our color detection algorithm to follow an orange track line on the circle. It was a difficult task to determine which portion of the camera reading images to run our color detection on without knowledge of neither the racecar's location on the track nor the track's shape when trying to minimize noise. In short, with the help of localization, we will be able to greatly improve the trajectory of our racecar in future implementations of similar path planning problems.

1.2 Overview

Throughout this lab, we were provided with and used a map of the Stata basement to create the environment for our robot to locate itself with respect to. We used LIDAR and odometry sensor readings as tools for determining the racecar's position and orientation. The main purpose of this lab was to learn to implement an accurate localization model using the given LIDAR and odometry data. The challenge was then to apply this localization model to determine the racecar's pose relative to the Stata basement during real-time driving.

In our solution, we utilized the Monte Carlo Localization (MCL) algorithm presented in lecture in order to determine the racecar's position and orientation in the map frame. MCL is a version of particle filtering, in which we use particles to represent the various different positions and orientations we predict the racecar could be at. We approached MCL by breaking it up into three parts: (1) Motion Model, (2) Sensor Model, and (3) Particle Filter. Motion Model utilized odometry messages to update the particle poses while Sensor Model utilized the LIDAR readings to determine the probability that each particle matched the racecar's location. Particle filter then provides the integration necessary for choosing which particles to update and how. The full implementation for each section as well as physical testing is detailed further below.

2 Technical Approach

2.1 Motion Model

Jeremiah Budiman

At the lowest level, the MCL algorithm works by utilizing a set of numerous particles. These particles' poses potentially match up with the pose of the actual robot. At each time step, each particle is translated and rotated based on the robot's odometry to acquire the new set of particle poses. This procedure is carried out by the motion model.

2.1.1 Robot Odometry and Noise

The robot's optometry is given as a 3 element vector $\begin{pmatrix} \Delta\text{x} \\ \Delta\text{y} \\ \theta \end{pmatrix}$ with the x component, y component, and angle difference component all being in the robot's frame.

A deterministic approach simply applies the given odometry to each particle's pose in the set to get the new poses. However, a more robust approach adds noise to the odometry. Noise is included because the particle poses may be generally accurate and around the same vicinity as the actual robot location; however, the accuracy of poses may increase if the robot were translated or rotated slightly wit added noise.

Thus, each particle in the set has an unique odometry. For each particle, the odometry applied is the given odometry plus random noise between the range -.1 to .1 added to each of the Δx, Δy, θ components.

2.1.2 Robot to World Frame

With the current procedure, our algorithm applies odometry with noise as intended. However, the odometry is in the local frame of the robot and must be converted to the world frame. To do so, the following rotation matrix is utilized:

$$\begin{bmatrix} \cos(\theta) & \text{-}\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Applying this rotation matrix to the (Δx + noise) and (Δy + noise) components, where θ also has noise, allows us to translate the particles to the world frame. Note that the angle difference in the robot frame is the same as the angle difference in the world frame, so no conversion matrix is needed for the orientation.

Finally, after acquiring the proper translation in the x-y plane and angle difference, each odometry with noise can be applied to the set of particles to get the new particle pose. These new particle poses can then be evaluated by the sensor model as discussed in the next section.

2.2 Sensor Model

Savva Morozov

The goal of this section is to develop a measurement probability model for our LIDAR sensor. This model allows us to evaluate the probability of which each individual particle would produce the obtained LIDAR measurements. Specifically, for a LIDAR scan $z_k^{1:M}$ attained at timestep k, our goal is to define the scan probability

$$p(z_k^{1:M}|x_k^p, m),$$

where x_k^p is the location of the p-th particle at timestep k, and m is the preset map of the environment.

2.2.1 LIDAR Scan Likelihood Model

In this section we express the laser scan probability model as a product of probabilities of individual range measurements in the scan. To do so, we assume that individual LIDAR scan-beams are independent (i.e., $p(z^i, z^j|x^p, m) = p(z^i|x^p, m)p(z^j|x^p, m)$) and equivalent (i.e., $p(z^i|x^p, m) = p(z^j|x^p, m)$). The independence assumption allows us to rewrite the scan likelihood model in a desired way:

$$p(z_k^{1:M}|x_k^p, m) = \prod_{i=1}^{M} p(z_k^i|x_k^p, m). \tag{1}$$

For visual clarity, we drop the timestep subscript in the following discussion. Given a particle location x^p in the map m, we can compute the ground truth distances $d^{1:M}$ that would have been observed by a perfect noise-less LIDAR scanner. This computation is performed using ray-casting — a procedure that finds the distance from the particle to the wall by projecting a beam from the particle. Ray-casting allows us to condition the range measurement model on the ground-truth laser distance, instead of the map and the particle's location:

$$p(z^i|x^p, m) = p(z^i|d^i).$$

Finally, since all the measurements are assumed to be equivalent, we note that we only need to define a single probability distribution $p(z|d)$ — the likelihood of obtaining a range measurement z if the ground truth distance is d. As we will see in the next subsection, this distribution is inherently nonlinear and may become computationally demanding when evaluated repeatedly. Instead, we approximate this continuous distribution with a discrete one; we also truncate both the ground-truth distance and the measurements to lie between 0 and $z_{\max}$. Such a discrete distribution can be computed in advance and stored in a lookup table for efficient evaluation. In the following section, we will define this discrete range measurement likelihood distribution.

2.2.2 Discrete Range Measurement Likelihood Model

As suggested by the course staff, we model the range measurement likelihood as a weighted sum of likelihoods that correspond to four independent events of interest:

1. A known obstacle is detected. The corresponding probability distribution is defined by a discrete truncated Gaussian centered at d:

$$p_{\text{hit}}(z|d) = \eta_{\text{hit}} \begin{cases} \exp\left(-\frac{(z-d)^2}{2\sigma^2}\right), & \text{if } 0 \le z \le z_{\max}, \\ 0, & \text{else}, \end{cases} \tag{2}$$

   where η_{hit} is the normalizing constant and σ is standard deviation of the Gaussian.

2. An erroneous short measurement is obtained. The distribution for this event is a downward slope:

$$p_{\text{short}}(z|d) = \eta_{\text{short}} \begin{cases} (1 - \frac{z}{d}), & \text{if } 0 \le z \le d \text{ and } d > 0, \\ 1, & \text{if } z = d = 0, \\ 0, & \text{else}, \end{cases} \tag{3}$$

   where η_{short} is the normalizing factor.

3. An erroneous large measurement is detected. The probability distribution for this event is a Kronecker delta:

$$p_{\text{rand}}(z|d) = \begin{cases} 1, & \text{if } z = z_{\max}, \\ 0, & \text{else} \end{cases} \tag{4}$$

4. A completely random measurement is received. The probability of this event is modeled with discrete uniform distribution:

$$p_{\text{rand}}(z|d) = \eta_{\text{rand}} \begin{cases} 1, & \text{if } 0 \leq z \leq z_{\max}, \\ 0, & \text{else} \end{cases} \tag{5}$$

The law of total probability now allows us to define the model on $p(z|d)$:

$$\begin{aligned} p(z|d) = &\, p(\text{hit})p(z|d, \text{hit}) + \\ &+ p(\text{short})p(z|d, \text{short}) + \\ &+ p(\max)p(z|d, \max) + \\ &+ p(\text{rand})p(z|d, \text{rand}) = \\ =& \alpha_{\text{hit}} p_{\text{hit}}(z|d) + \alpha_{\text{short}} p_{\text{short}}(z|d) + \alpha_{\max} p_{\max}(z|d) + \alpha_{\text{rand}} p_{\text{rand}}(z|d), \end{aligned} \tag{6}$$

where $\alpha_{(\cdot)}$ are the priors on individual events — weights that sum up to 1. This range measurement likelihood model $p(z|d)$ is the discrete probability distribution that is encoded into the lookup table. This allows us to compute the probability of a particle producing a particular laser scan as in (6), which will be used as an important weight in the particle filter.

2.3 Particle Filter

Miranda Cai

In this section, we describe how we used our motion model and sensor model to implement an accurate MCL particle filter algorithm that localizes the robot.

2.3.1 Particle Initialization

As previously mentioned, particle filtering works by initializing and continuously updating a number of N different particles by their position and orientation that could each represent the racecar's true pose on the map. To first initialize the pose of our particles, we subscribed to a ROS topic that retrieved the current starting pose of the racecar immediately upon starting our algorithm. Then, we created N particles that were all relatively close to this pose by injecting a random amount of noise within a specific range to this pose N separate times. Adding random noise accounted for any uncertainty within the odometry message we received for the racecar's initial pose such that there would be a higher chance that at least one of the particles contains the true pose of the racecar when the particles are different.

Although we could have instead simply initialized the N particles completely randomly across the map, we chose to randomize them within the scope of the racecar's initial pose for efficiency. The map of the Stata basement was relatively large; thus if we initialized the particles across its entirety, we most likely would have had to go through numerous additional particle updates before the particles neared the racecar's actual location - a process that takes extensive time.

Additionally, in our specific design for the initialization, we decided to use $N = 200$ particles throughout. The thought process behind this number was that we wanted N to be as large as possible such that the noise in our model approaches 0 as quickly as possible to imitate a deterministic model while considering the limiting factor of efficiency. Through trial and error, we found that 200 was enough particles to reach this desired accuracy and not too computationally heavy to still be able to run in real time on the racecar.

2.3.2 Particle Updating

After initializing the particles, we start to update them to become closer and closer to the racecar's actual pose. We update the data by subscribing to our LIDAR and odometry topics, and update the particles during our subscription callback functions as described below.

Whenever we receive motion readings in the form of odometry messages, we make a callback to our implementation of the motion model with the current particles and new odometry message as inputs. Motion model then outputs the new particles poses, which are calculated to take into account the movement in our car from its previous location.

Similarly, whenever we receive sensor readings in the form of LIDAR scans, we make a callback to our implementation of the sensor model with the current particles and new LIDAR scan as inputs. Sensor model then calculates the probabilities that each particle pose matches the actual racecar pose. From here, we use these probabilities to gradually filter out poses that are unlikely to be our racecar. We execute this filtering process by first normalizing the probabilities of particles we just found so that the sum of all particle probabilities equal 1. Then, based on this normalized probability distribution we resample a new particle N times to get our new particles.

In theory, after each update in this manner, the particles should first get closer to each other because we are filtering out and only keeping a portion of our old particles (where particles with high probabilities will get resampled more often than particles with low probabilities). Second, each update should bring the particles match the racecar's actual pose more closely because the racecar's exact LIDAR scan should only be producible from that specific pose.

It is important to note that in very symmetrical environments (such as square rooms), relying on LIDAR scans can prove difficult since certain poses will output the same LIDAR scans. However, in this lab, we only needed to test on the Stata basement which is very irregular in shape all around. Due to this irregularity, distinct poses should provide a unique LIDAR scan, which is why using LIDAR scans as our sensor data was sufficient in this case.

A second note is that our team decided to implement our update steps using safe threading techniques. Since our LIDAR and odometry callbacks were independent, our updates could have conflicted by having one method begin updating the particle poses before the other method completed its own update. To counter this possibility, we initialized a mutex object at the start of our code. Then, any time we needed to access/edit the particle poses, we set the thread to acquire the mutex right before and release it right after.

2.3.3 Getting the Localization

After successfully initializing and updating the particles, the final step of our localization model is translating the N particle pose into the racecar pose belief. We do so by taking the "average" pose out of all particles after each particle probability update (calculated after receiving a LIDAR message as described above), and setting this value to be the racecar's pose.

For the x, y, and θ coordinates of the particles, we take the values with the greatest probability for each and combine them to use for our "average" pose. We decided to set the pose of the car this way to account for the possibility that our probability distribution is multimodal, in which case taking an arithmetic mean could end up returning a particle pose that is between modes, such that its own probability of matching the car's pose is low. By selecting each coordinate by maximizing its probability, we are essentially taking the "best" particle representation rather than just an average.

For visualization purposes, we publish the average pose in the form of an odometry message and a transform broadcast. This way, we can see definitively whether or not our calculated pose is at the racecar's actual location, which demonstrated to be especially helpful for debugging during the hardware implementation.

2.3.4 ROS Implementation

The implementation of particle filter onto the physical racecar follows in the exact same method, with a few adjustments to parameters and constants.

3 Experimental Evaluation

3.1 Quantitative Analysis

Bill Kuhl

3.1.1 Simulated Testing

Given our resources, we focused the quantitative analysis of our motion model, sensor model, and particle filter in simulation. In simulation it is possible to obtain the exact ground truth of the vehicle to compare to our results. To get the same ground truth when operating in hardware, a secondary method of tracking the robot with high precision would have been necessary.

Analysis using the simulation was conducted in two parts. The first was a qualitative analysis and debugging of published odometry data from the particle filter. The second was a numerical analysis comparing the results of our model with the ground truth and the TA's solution, conducted on Gradescope. To complete the qualitative analysis, we ran the provided racecar simulator package in ROS, concurrently running our localization package. Then, the car was put in motion using the wall-follower algorithm. By plotting the published odometry data in the RViz terminal, it was possible to see if the desired behavior was being shown. In this case, the desired behavior was the odometry data centered at the body of the car with the direction pointing orthogonal to the front of the car. Early tests and iterations of our localization algorithm produced data which was non-deterministic and became more erratic as time went on. This prompted us to debug and create an algorithm that, in the end, produced qualitatively desirable behavior in these circumstances. This behavior can be seen in Figure 2, with Figure 1 showing the 'messy' original localization data. Quantitative results were generated by running a similar odometry problem in Gradescope and comparing our results to the TA solution to generate numerical data. An analysis of these results can be found in the next section.

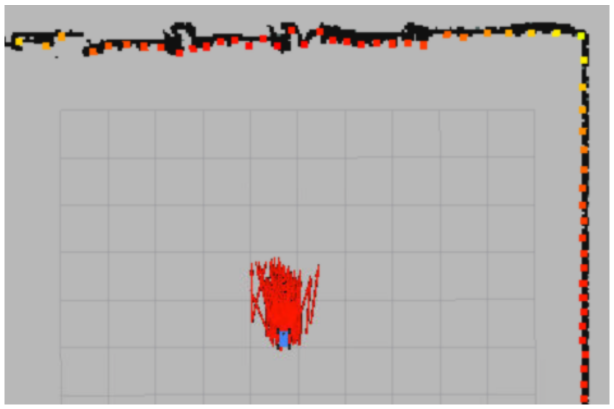

Figure 1: Messy localization at initialization.

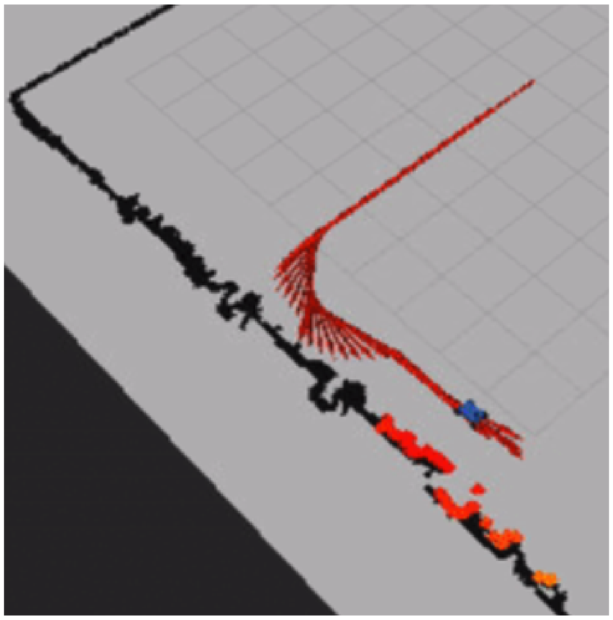

Figure 2: Successful localization over time.

3.1.2 Results

Results in simulation were based off of the distance to the desired path for odometry data. This time-averaged error over the entire simulation was compared with the staff solution, as graphically shown in Figure 3, where the red path represents our solution, the green represents the staff solution and the blue represents the ground truth.

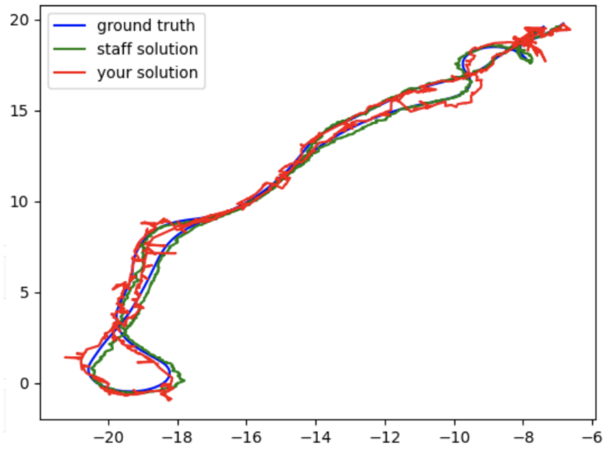

Figure 3: Graphical representation of Gradescope autograder result.

The motivation behind comparing our solution with the staff solution was to determine what a reasonable deviation from the ground truth would be. There will be errors present in the solution inherently because localization is an estimation, so such analysis with a known suitable result is important. This comparison with the TA results is measured by a comparison score metric defined as follows:

$$ComparisonScore = max(0\%, 100\% - 1/m * max(0, OurError - StaffError)) \tag{7}$$

Because the data we create our localization estimation on can have an indeterminate level of noise from the sensor, it is important to test our model with various levels of noise. We conducted the same test comparing our results to the ground truth and the TA results in three different levels of odometry noise: no noise, some noise, and more odometry noise. It would be useful to know the actual level of noise, however this is obfuscated. The results can be found in Table 1.

Table 1: Quantitative results of simulated testing (Gradescope)

Noise Level	Time Average Deviation from Target	Comparison Score
None	0.22m	0.98
Some	0.31m	0.90
More	0.29	0.93

These results indicate that while our system does a comparable job producing odometries similar to the target, it is not as stable to noise as it could be. Based on these results, future work could be spent on making the results more resilient to noise.

3.2 Qualitative Analysis

Tiasa Kim

3.2.1 Hardware Testing

After testing our particle filter in simulation, we tested it on hardware. Our preliminary setup for hardware testing included adjusting the node topics to publish on the racecar; we changed the transform of the particle filter frame from /base_link_pf to /base_link and the /odom topic to /vesc/odom. Additionally, we set the Stata basement as our testing environment (given by the map image in the lab files).

Our performance metric for hardware testing involved a qualitative assessment based on human judgment of how close in approximation the racecar's location in the real environment is to the visualized LIDAR scans and particle positions. To evaluate, we used RViz as our main tool for visualizing both the simulated environment and published LIDAR and odometry data. The odom, scan, pose and map topics, as well as the robot model, were added in RViz as visual markers for interpreting the racecar's location and movement in the environment. Additionally, we published the average pose as an odometry and a transform broadcast message. Publishing this data helped visualize and further evaluate the correctness of the computed pose.

Experimenting on the physical racecar allowed for a more comprehensive evaluation. First, it added a layer of qualitative analysis through an integrated human factor and visualization tool method. Furthermore, because there exist limitations with simulations (i.e. there are certain variables in a real environment, such as unpredictable noise, that cannot be anticipated and shown in complete representation in simulation), hardware testing helped identify discrepancies that might not have been found during simulated testing. The discrepancies found in our testing are discussed in the following Results section.

3.2.2 Results

During our initial phase of hardware testing, we ran into a couple issues, one of which was an incorrect initialization of particles. When initializing the particle distributions in RViz, the estimated location of the simulated racecar was offset by approximately 5 meters ahead from the actual location of the racecar. This offset is shown in Figure 4, where p_{actual} represents the racecar's real location and p_{viz} represents the location in RViz.
Additionally, when manually driving the racecar forward, the updated poses of particles were published with a lag (as displayed in RViz) compared to the real time updates of the racecar's actual position. Furthermore, the particles were often published in the wrong orientation and localized with noisy data, as seen in Figure 5.

However, with additional debugging with the particle filter and retesting on hardware, we significantly improved (per qualitative assessment) our particle localization.

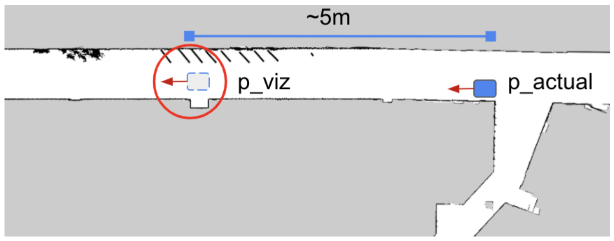

Figure 4: False initialization of set of particles as represented by robot location.

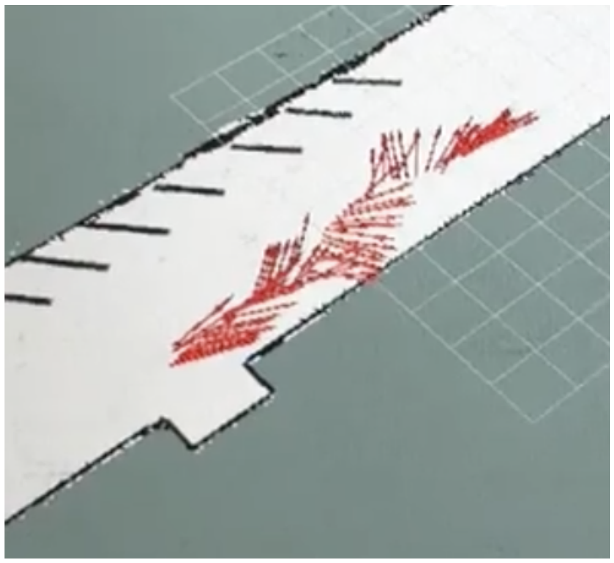

Figure 5: Pose published in backwards direction and skewed by noisy data.

Fixing these issues was a two-fold approach of optimized engineering and better understanding of theory. From an engineering approach, we fixed the particle initialization problem by implementing a separate publisher and subscriber node (using TA-provided scripts) and hard-coded the initial pose of particles. To improve the overall speed and accuracy of our particle filter, we rewrote our sensor model to evaluate at 100 times faster. Finally, closer debugging led to finding a bug in our subscribed topics method such that the motion model was never actually called on the hardware tests. Based on further exploration of concepts, we significantly reduced the noise covariance, which was previously too high for the time-scale of 10 m/sec. We learned that estimating the uncertainty of the system is an important step. An too-high estimation results in artificial and unnecessary noise injection; whereas, a too-low estimation results in a deterministic model. In addition, we used the built-in function *argmax* of distribution - not expectation (i.e. mean). This approach of using maximum likelihood worked better because the distribution over state may be multi-modal. Lastly, we switched to adding Gaussian noise instead of uniform noise. Gaussian noise demonstrated a more robust outcome because it allows us to calculate probabilities by using mean and standard deviation to determine the likely outcome above and between values. Uniform distribution, on the other hand, does not use a mean value; thus this method results in an equal (i.e. uniform) likelihood between any two boundary values.

The improved cleanliness, speed and accuracy of our published particles can be seen in the following two external videos displaying RViz visualization during hardware testing: Video demo of improved particle filter 1 and Video demo of improved particle filter 2.

4 Conclusion

Tiasa Kim

This lab was a multi-stage solution approach to the localization problem, where the objective was to estimate the robot's position and orientation (i.e. pose) in a given environment. We successfully implemented a motion model and sensor model, which allowed us to integrate the two models into our implementation of the particle filter. Our particle filter implementation sufficiently passed the auto-graded Gradescope tests and showed robust localization of particles during simulated testing. A few critical issues occurred during hardware testing (which did not occur during simulated testing). These issues were important to debug and fix since localization is an essential component of implementing a robust path-planning algorithm (as part of our future lab work and final race challenge).

Moving forward, we hope to apply our technical and theoretical lessons learned as we work on implementing an optimal path-finding motion-planning algorithm in Lab 6. From our experience with in this lab, multiple eyes for reviewing code and debugging, as well as a stronger understanding of why and how we implement certain parts of a model or algorithm, proved to be critical for success in robotics / engineering-related work.

5 Lessons Learned

Jeremiah Budiman

There were many things that were learned from this lab. From a technical perspective, a large portion of the lab was dedicated to debugging. Most times, these debugging sessions took several hours and were resolved with quick one line or one variable fixes. The key things to learn from this experience is to start early, giving ample time to resolve issues like this, as well as have peers check over your work if the code still fails; often times these one line bugs were fixed after multiple people would take a look at the code, often offering their solutions based off their own perspective.

From a CI-M perspective, I learned that not everybody has to physically be in the same place to have a productive work session. Due to spring break, members from our team were scattered in different places, making it impossible to meet up in person. However, we did a great job of communicating and produced a viable solution to the problem. This required constant communication, usually in the form of text message.

Miranda Cai

In terms of CI-M for this lab, I learned that trying to have 4 or more people collaborate on one file at a time is difficult to do without making at least one person feel left out. For example, we all came together to write up particle_filter.py, however soon enough realized that it was near impossible to keep everyone up to date with what was happening on one person's computer screen all at the same time. We then all tried testing different methods simultaneously, but doing so just resulted in a ton of merge conflicts or overwriting of someone else's ideas, which looking back was neither efficient nor optimal. A better way to do this in the future would be to divide up the work even more so in the beginning, and in the case where everyone wishes to help out on a part we could instead all brainstorm ideas before touching the code, and then assigning only one or two people to implement them so not to waste everyone's time in the future.

For the technical part of this lab, I reencountered that debugging will take longer than expected as always, and most of the time be because of something we didn't even consider to be a bug. For example, when we were getting a really low accuracy for our model, we assumed there was something wrong with our averaging or update step and spent hours looking for it there. It turns out it had to do with our particle initialization however, a part of the code so short and straightforward that we took for granted to be correct. A better way to debug would be to scan through every piece of code before going into depth which is just a small portion and doing everything to fix something that wasn't even broken.

Tiasa Kim

One lesson I learned from experience in this lab was the importance of attending to built-in function details. For example, we found out that we should use numpy rather than scipy to compute the circular mean of poses. This was the case because scipy assumes that the angle range is 0 to 2pi rather than -2pi to 2pi to account for negative angle values as applicable. Similarly, I realized through the hard way that debugging can take a rather long time and sometimes end up solving a very small, one-line mistake. This tricky nature of debugging makes programming a practice of critical thinking / problem-solving, as well as a skill of detail-orientation.

In addition, working on this lab reminded me that individuals function differently in terms of when and how one processes information and concentrates. From talking to one of my other teammates, I learned that (similar to me) this person comprehends concepts better when grappling with the material alone rather than out loud with or in the presence of others. As for myself, I find it harder to process information when others are surrounding me, which is most likely explained by the pressure felt from needing to understand the concept quickly to match the rate or expectations of others. This individualized learning experience makes teamwork challenging because it forces us to be more understanding of individual differences, likewise individual backgrounds that might not always be laid out in front. Lastly, I have realized again how important sleep is and how a lack of quality sleep messes up with one's brain functionality.

Bill Kuhl

Technically, I learned quite a bit about the importance of good documentation and unit definitions. I know when we were working on the particle filter there was a moment where I had written a subroutine, only to realize at the end while I had written everything correctly, the desired output was unclear. This was solved because everyone was sitting together and we could talk about what we needed, but it would have been problematic had we been working asynchronously. Additionally, because we did not pay attention to units we spent about 5 hours working on solving a bug that was because a scipy function we were using went from $0 - 2\pi$ instead of $-\pi - \pi$.

Working in the group I learned quite a bit about how to manage work in bigger groups. It is difficult to work on one code file at the same time among five people, so it was necessary for us to break up the work so we could all work separately and then come together for integration where we each could pair up to work on separate sections of the project. I think that maybe we could have started to work earlier; however with spring break and all, I don't think anyone was in the mood at all (rest is super needed). We also need to distribute the workload of who specializes in what. I realized this week we have like 2 people who are the hardware experts and the rest of us would struggle a little bit working with the physical thing itself.

Savva Morozov

I find the lack of theoretical rigor in the problem assignment inappropriate. Throughout the assignment, I found particular variables and terms to be outright undefined (map scaling and resolution), important assumptions unmentioned (range scan independence and equivalence), and theoretical concepts — unjustified and brushed under the rug (law of total probability on events).
We are at MIT, we can do better. We the students can handle the math. Robotics is all math. We are covering extremely complex concepts, but instead of diving into the theory, it feels like hacking, as if we are doing things for the sake of a nice-looking demo.
Granted, we are solving extremely complex problems. Defining them fully would take a lot of effort, and I understand that there are bugs and errors. What frustrates me is that these bugs are caused by lack of rigor. As a result, we have incorrect Gradescope tests, even after a fix. I spent hours debugging my correct code so that it could pass incorrect Gradescope tests.

So what did I learn? I was reminded that there are two ways to do work. First is shallow: read through the assignment diagonally, start hacking, debugging, nothing works, I get frustrated, start hacking up some solutions. This takes 5-10 hours until anything works at all. Or I can spend 1 hour of deep effortful focus to understand the theory, and another hour to code everything up. Two hours total. Deep focus and rigor is hard. Understanding the theory is hard. But it's the right way, the only way.