Lab 5 Report: Localization

Team 23

Alazar Lemma
Benjamin Soria
Caden Moore
Henry Heiberger
Vittal Thirumalai

Report edited by Caden Moore

Robotics: Science and Systems

April 15, 2023

1 Introduction

Authored by Alazar Lemma, edited by Caden Moore

As a team, we have achieved a considerable amount of progress in creating an autonomous racecar by learning how to implement wall following and visual servoing on the robot and pushing through the challenges that we encountered while doing so. However, for us to be able to compete in the final competition in May, we must also implement localization. Robot localization, or the robot's ability to determine its location on a map, is a relevant guide to navigation as it allows the robot to implement higher level planning, and grants the robot the ability to make decisions in the future. Without localization, our robot is essentially driving blind without the ability to navigate complex environments in the absence of human intervention. In combination with all previous labs, this further enhances our robot's autonomous capability, giving it a way of knowing where it is in the world and ultimately setting the foundation for path planning.

In this lab we were tasked with the goal of developing a localization method for our autonomous racecar and our group approached this problem using Monte Carlo Localization (MCL). MCL is a probabilistic localization method that maintains many particle estimates of the robot's location and estimates the location of the robot based on the probabilities of each particle being the robot's actual location.

To achieve a working MCL algorithm, we took an incrementally progressive approach and first implemented a motion model which was designed to update the position of particles with some added noise. Then we implemented a sensor model which calculated the probability of each particle being the location of the robot. With the motion and sensor models, we then were able to create a particle filter that uses these probabilities to choose the most likely position of the car and keeps updating the estimated position of the robot. We then ran tests in simulation and checked our results with the autograder, and upon proving that the simulation was sufficient, we were able to implement the model on the robot. In addition, our group went ahead and implemented simultaneous localization and mapping (SLAM) using Google Cartographer in simulation. This report will illustrate our journey to implementing localization onto the racecar, bringing us one step closer to a fully autonomous robot ready to compete in the competition in May.

2 Technical Approach

2.1 Implementing Monte Carlo Localization

Authored by Henry Heiberger, edited by Caden Moore

Localization is the process of algorithmically determining and maintaining where a robot is located with respect to its environment was the primary purpose of this lab. Specifically, our technical problem was to implement Monte Carlo Localization (MCL) which is a probabilistic localization algorithm that estimates the position of the robot through the use of a large particle distribution of likely states. By comparing the ray cast simulation of LIDAR data of each particle position estimate to the actual LIDAR data of our racecar using a probabilistic model, we were able to compute the likelihood that each particle correctly estimated the ground truth location of our vehicle and continuously resample our particles to produce a better guess. This section highlights and rationalizes our technical approach to solving this MCL problem as well as discusses its implementation in depth.

2.1.1 Part 1: The Motion Model

The first major implementation step of our MCL algorithm was implementing the motion model. In essence, the Motion Model is a step that takes in the odometry data of the robot and uses it to update each of the MCL particles' position based on its previous state. This can be done by applying the definition of the 2D pose matrix and making use of pose composition through the simple relationship shown in Equation 1 where P_k^W represents the pose matrix of the robot with respect to the world during the current time step, P_{k-1}^W represents the pose matrix of the robot with respect to the world during the previous time step, and ΔP represents our odometry converted into the same 2D pose matrix form.

$$P_k^W = P_{k-1}^W * \Delta P \text{ where we represent our 2D pose matrix in the form } \begin{bmatrix} cos(\theta) & -sin(\theta) & x \\ sin(\theta) & cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

It is worth noting that, to maximize the above calculation's efficiency, we compute this motion model update over a vector of all of our particles in a vectorized fashion by directly applying the formula found when expanding the matrix multiplication shown in equation 1 to each particle row instead of converting the robot pose and odometry into its 2D pose matrix representation during each update step.

While the above motion model would work perfectly in an ideal simulation environment, the odometry recorded by our robot in the real world is almost certainly noisy, meaning we cannot rely on it to perfectly reflect the true motion of our robot. To account for this, we adjusted our motion model to more accurately reflect real-world noise by adding a normal distribution of noise centered on 0 to the odometry of every particle. A pictorial example of our motion model acting on a single particle with and without added noise can be seen in **Figure 1**.

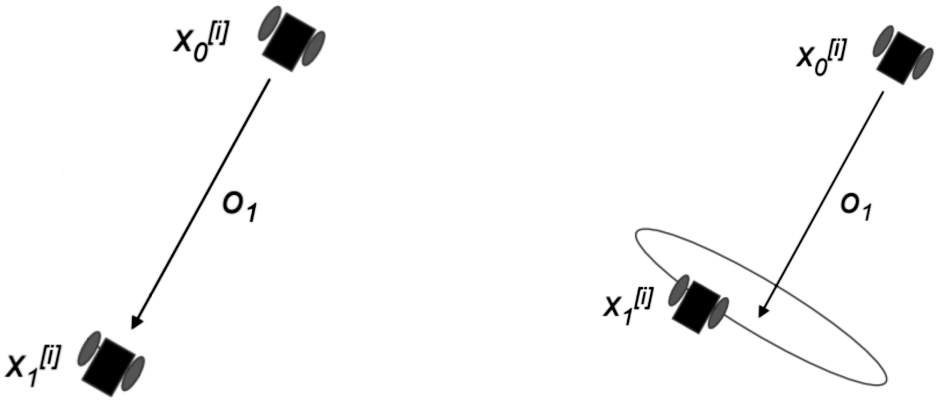

Figure 1: The left image shows the effects of the motion model on particle i when no noise has been added to odometry o_1. The right image shows the case where odometry noise has been added, giving the robot a range of locations in which it could arrive at.

The scale (modified through the standard deviation of the normal distribution) of this added noise was set through extensive testing both in simulation and on the robot. We noticed that making the standard deviation of the noise too small led to our particles being unable to successfully adjust when world noise entered the robot's odometry, leading to drift after extended localization tests, and that making the standard deviation too large led to an unstable robot pose estimate. Thus, we ended up settling on a middle ground value of 0.02, producing a motion model that allowed our MCL algorithm to perform well both in simulation and on the robot.

2.1.2 Part 2: The Sensor Model

After our motion model the next major section we implemented for our MCL algorithm was the sensor model. The sensor model is the module that takes in our Robot's LIDAR data and uses it to calculate the likelihood of each particle being in the same position as the robot. It does this by first taking ray casts on the map from each particle location in order to simulate measured LIDAR data and then applying the probabilistic model described by Equation 2 on each individual data sample where $\alpha_{hit} = 0.74$, $\alpha_{short} = 0.07$, $\alpha_{max} = 0.07$, and $\alpha_{rand} = 0.12$

$$p(z_k^{(i)}|x_k, m) = \alpha_{hit} \cdot p_{hit}(z_k^{(i)}|x_k, m) + \alpha_{short} \cdot p_{short}(z_k^{(i)}|x_k, m) + \alpha_{max} \cdot p_{max}(z_k^{(i)}|x_k, m) + \alpha_{rand} \cdot p_{rand}(z_k^{(i)}|x_k, m) \tag{2}$$

However, in order for MCL to localize effectively, our algorithm must operate with a large number of particles and at a high clock rate. This makes it computationally infeasible to have to compute the above equation explicitly for each particle every cycle. To solve this problem we discretized the range values of our LIDAR and ray cast data into 200 potential buckets and precomputed our probability values into a 200 x 200 lookup table. This allowed us to only have to reference the values in the table each time we wanted to draw probabilities from our sensor model. To maximize efficiency, this table lookup was done in an efficient vectorized manner through the use of Python's Numpy library. A visual representation of this precomputed sensor model table can be seen in **Figure 2**.

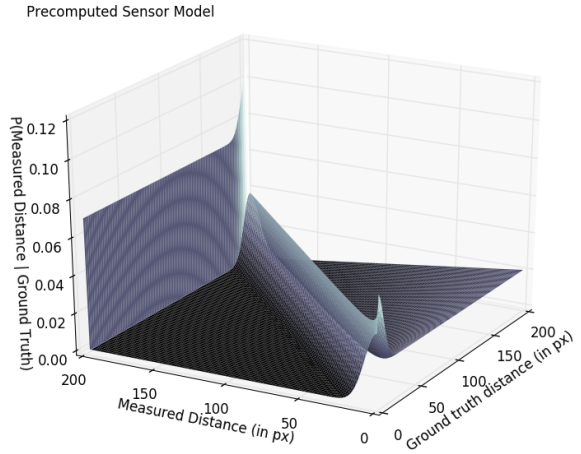

Figure 2: This graph shows a visual representation of our precomputed sensor model table. It shows the probability that a particle represents the true robot position when given the measured distance of a particle taken from a ray cast and the ground truth distance taken from the real robot's LIDAR sensor.

We also note that before returning our list of probabilities we found the full probability for each particle by aggregating the probabilities found for each ray cast sample taken through multiplication and then "squashed" our sensor model output by raising it to the power of $\frac{1}{2.2}$. This was done to make the probability distribution returned by our model less peaked, increasing the likelihood that particles close to the true location of the robot receive significant probability scores. If the distribution was instead too peaked, there is a higher probability that even a particle that very closely represents the position of the robot could still miss the peak and receive a lower probability than it should.

2.1.3 Part 3: The Particle Filter

Once our motion model and sensor model were complete, our final task was to put the two together to implement our overall particle filter. Before this could be done though, there were a few final design decisions that needed to be made.

The first major design decision was to initialize the particles of our MCL algorithm. In an ideal solution, we knew that the algorithm would perform best if the particles were initialized at exactly the ground truth position of the robot (this is because this essentially eliminates the need for initial convergence at the robot's position). However, similar to our argument regarding our robot's odometry, when performing localization, there can be some uncertainty about where exactly the robot starts on our map. Thus, we initialized our particles through a normal distribution centered on our best estimate of the true starting location of the robot (in simulation, this estimate was exact). This was done to ensure that even if we were somewhat off with our initial position estimate, there was still a high likelihood that some MCL particles would still start very near the robot's starting position, giving them a high probability in the sensor model and causing them to be favored during resampling. The exact scale (or standard deviation) of this normal distribution was again chosen through experimentation but could be altered for situations in which there was a higher degree of uncertainty in our robot's starting position. In most informed situations though, we found a standard deviation of 0.5 to perform well through quick and accurate particle convergence. An example visualization of our particle initialization in simulation can be seen in **Figure 3**.

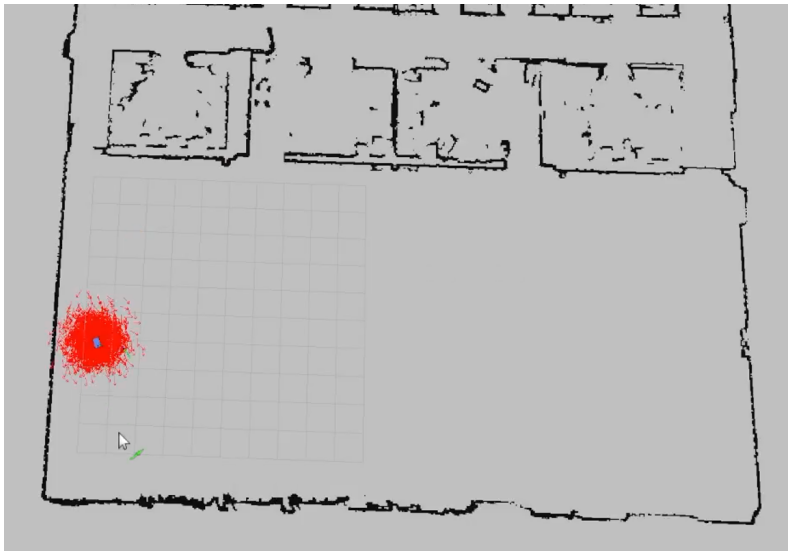

Figure 3: An example of particle initialization within a simulated MCL environment. The blue model indicates our simulated robot racecar and the many red arrows each indicate a random initialization of a particle. In further time steps, the particles will quickly converge on the robot, giving us an accurate estimate of its pose

Beyond particle initialization, we also deemed that it would be inefficient to process all 1000 LIDAR beams collected by our robot within our MCL algorithm. This is especially true because doing so would require every particle to take that many ray casts during every sensor model update step. Thus, before passing our LIDAR data into our sensor model, we first downsampled it into only 100 evenly spaced samples. This number was chosen primarily for efficiency purposes as we found through extensive testing that increasing this number did not substantially affect the accuracy of our localization but it did substantially affect the rate at which the algorithm performed.

Once these two final design decisions were complete, the final step of implementing our overall MCL algorithm was to assign ROS subscribers that triggered a motion model update callback whenever new odometry data was received and a sensor model update callback whenever a new LIDAR scan was taken. In addition, whenever the sensor model returned a new list of particle probabilities, we resampled the particles with a probability distribution that followed the sensor model's output. A pictorial representation of this final MCL process can be seen in **Figure 4**.

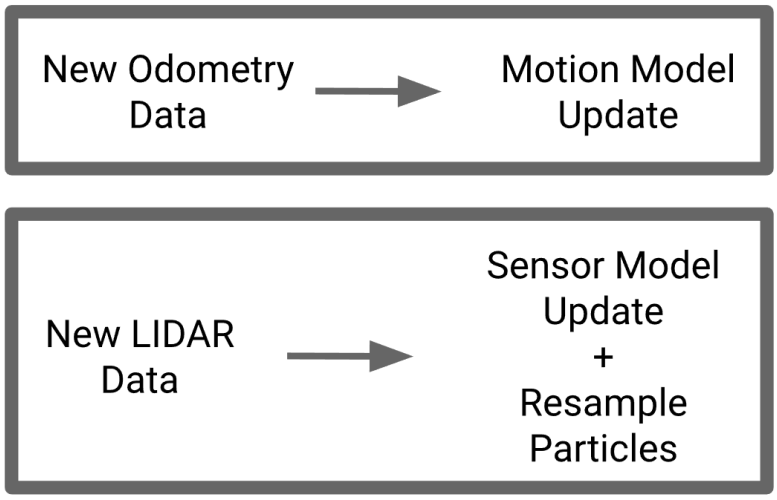

Figure 4: This figure highlights the two major processes running by our MCL algorithm. It shows how, whenever new odometry data is received by its subscriber, a motion model update is triggered. In addition, whenever new LIDAR data is received by its subscriber, a sensor model update is triggered and the particles are resampled based on the newly computed probabilities.

It is also worth noting that because our array of particles is shared between both our asynchronous motion model and sensor model callback updates, care had to be taken to ensure that both models didn't try to update the array at the same time. To solve this problem, we applied mutual exclusion by adding a lock that locked down the global particles array whenever either model attempted to access it. Though the contention of this surely had some impact on the performance of our MCL algorithm, we concluded that it was necessary to make use of this lock in order to ensure our sensor and motion models always output the correct results.

Following all of these steps, we achieved an implementation of the MCL algorithm that accurately identified and tracked the position of our robot both in simulation and the real world. A visual example of this algorithm in action can be seen in **Figure 5**.

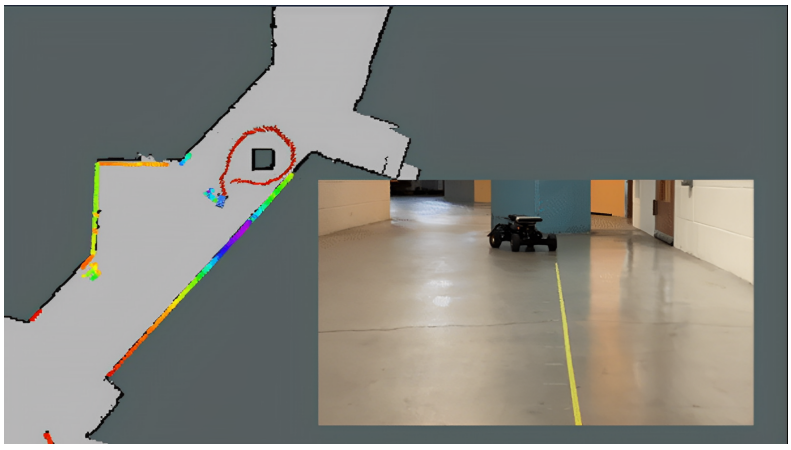

Figure 5: An example screenshot taken while testing our MCL on a live robot. The inner image displays a live recording of the robot driving while the outer image shows the map on which the robot is localizing. Within the map, the red path indicates where the robot has come from and the rainbow lights represent data taken from the robot's LIDAR sensor. It is worth noting how great localization performance can be verified both by comparing the location of the robot on the map with the location of the robot in the environment and by how the LIDAR data taken from the robot lines up nearly perfectly with the walls of the map.

2.2 Implementing SLAM through Google Cartographer

Authored by Vittal Thirumalai, edited by Caden Moore

After successfully implementing MCL, we attempted to deploy an implementation of simultaneous localization and mapping (SLAM) on our simulated racecar environment through the use of Google Cartographer. This section describes our experience and results when working with this open-source software system.

To deploy SLAM on our simulation through Google Cartographer we had to first install the Cartographer packages. This essentially required us to just follow the provided documentation. However, during this process there were several bugs and issues with importing the libraries that had to be solved. One particularly noteworthy example occurred when we forgot to run "docker compose down". This caused numerous bugs to pop up when running the cartographer on the racecar simulation causing us to end up just removing the Cartographer folder and restarting the installation process. Eventually, though, we were able to debug all the library and import issues we faced.

Once installed, to actually run SLAM through Google Cartographer we ran three files in parallel: the racecar simulator to manage our simulated racecar environment, the Cartographer to perform SLAM dynamic localization, and an extra node to allow us to move the car around and discover the map. To visualize this mapping process in RViz, we added submaps on the topic /submap_list and map on the topic /cartographer_map. With this, we were able to watch how, as our robot drove around the simulation environment, it was able to generate its own accurate map of the space in real-time. An example of Google Cartographer running in progress can be seen in **Figure 6**.

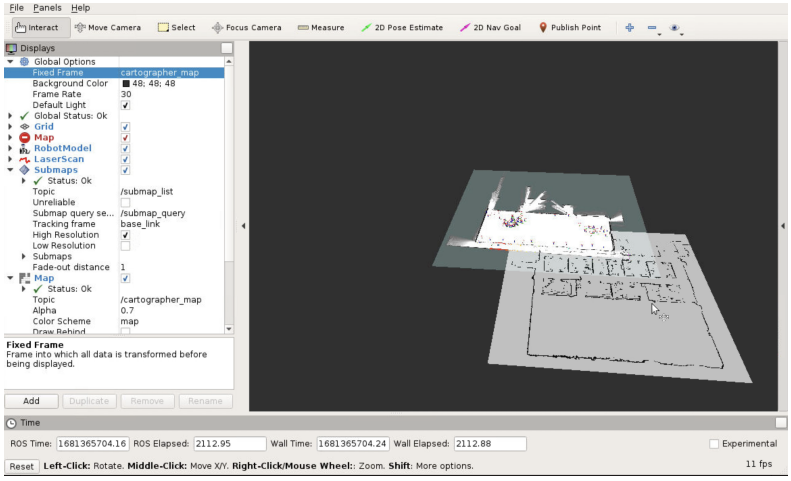

Figure 6: A screenshot showcasing our RViz with Google Cartographer operating in progress. Note how the car's exploration of the environment has already allowed it to map much of the large rectangle space of the simulation map and adjacent corridors.

To allow our robot to discover the whole simulation space, we used the parking simulator from Lab 4. This simulator allowed us to click on a goal point and have the car travel to it using a pure pursuit. By doing this repeatedly and making the car travel into all of the small pathways of our simulation map, the LIDAR on the simulated car was able to detect all the walls and thus allowing the Cartographer to fully generate an accurate portrayal of the simulation map.

3 Experimental Evaluation

Authored by Benjamin Soria and Caden Moore

The experimental evaluation of our implementation took advantage of three tools available to our team: simulation, auto-grading, and real-life performance. It was necessary to rely on tests performed in simulation and auto-grading because the nature of the problem requires us to incorporate uncertainty in several stages (applying normally distributed noise to the transformation applied on the particles, resampling with probabilities, and initializing particles with a normal distribution). Collecting data in the real world is valuable and necessary but introduces noise that can't be accounted for. It is difficult to compare the performance of two runs executed with the same parameters and initialized in the same manner due to differences in LIDAR beam, runtime, or wireless communication.

3.1 Simulation

Authored by Benjamin Soria

The motion model attempts to accurately reflect real-world noise by incorporating a normal distribution of noise centered on 0 to the odometry of every particle. A standard deviation governs the span of noise added. Figure 7 shows the distribution of particles with the standard deviation set to 1.0 and 0.02.

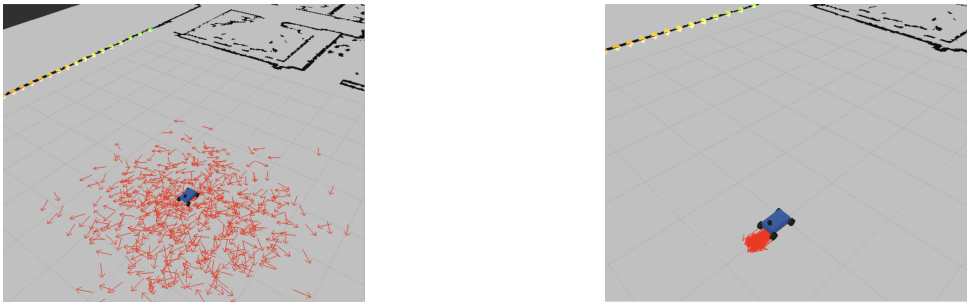

Figure 7: The left image shows the effects of setting a standard deviation of 1.0 to the normal distribution of noise implemented on the odometry of every particle. The right image shows the effects of setting the standard deviation parameter to 0.02.

Six tests were performed in which the standard deviation varied from 0.02, 0.10, 0.25, 0.5, 0.75, and 1.0. To compare our estimate of the odometry, which is the average (x, y, θ) value, to the ground truth, we set the position of the robot to $(x, y, \theta) = (0, 0, 0)$, then, the particle motion was launched and a rosbag was started to record the average (x, y, θ) value. After a few seconds of recording, the initial position was set to $(x, y, \theta) = (1, 1, 0)$. This sudden change in position simulates a step input, such as starting the car in a new environment. Figure 8 displays the average y-particle value for a given noise parameter, indicating our estimated odometry of the y coordinate. An ideal controller would converge from $(0, 0, 0)$ to $(1, 1, 0)$ with a small convergence rate. To measure the convergence rate, we measured the time required for the estimated odometry to transition from the starting to the end point; the collected data is shown in Figure 9.

We observed that increasing the noise parameter introduced greater uncertainty in our estimate of our current odometry. Picking a small noise parameter value, such as 0.02 resulted in a precise, however, inaccurately estimated odometry of 1.2 m. A noise parameter of 0.10 meters seems to provide the best combination of precision and accuracy. Beyond a noise parameter of 0.10 meters, the estimated y-value varied to much to result in a precise controller. It is worth noting, however, that even with a high noise parameter, the odometry still converges to the ground truth value of y = 1.

The convergence rate seems to decrease as the noise parameter is increased. Figure 8 and Figure 9 agree that higher noise parameters do result in faster convergence to the ground truth. It is worth noting that beyond a noise parameter of 0.75, convergence is almost instantaneous. This may be the result of a sampling error introduced by a the speed at which we are able to subscribe, process, and publish data.

Throughout development in the real world, we tested at the extremes of this parameter (0.02, 0.5, and 1.0) and decided that 0.02 would be the noise parameter moving forward. Given more time and the results of analyzing our odometry in simulation, we would like to further explore how a noise parameter of 0.10 performs in the real world. Our data indicates that this parameter results in a suitable convergence time of about 1 nanosecond and is precise and accurate. Though we do not expect that any increase in real-world performance gained by modifying this noise parameter will noticeably improve localization ability.

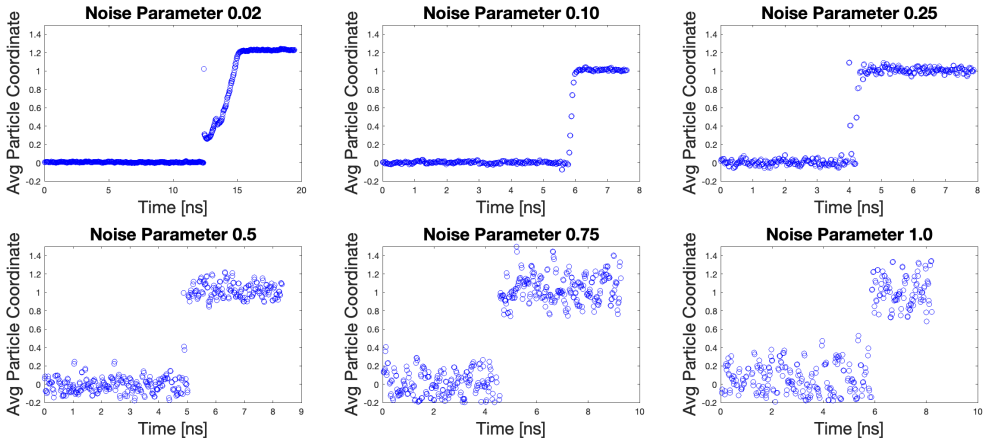

Figure 8: The two plots above show the average-y particle location for noise parameter incorporating 0.02 and 1.0 standard deviations from the mean, respectively. The bottom two plots show a running average with a width of 20 incorporated to smooth out data and calculate convergence time.

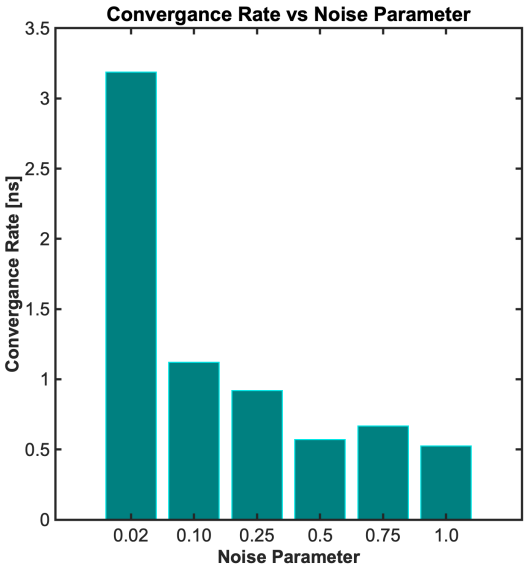

Figure 9: The convergence rate for each noise parameter tuning was calculated as the elapsed time between the transition from a position of $(x, y, \theta) = (0, 0, 0)$ to $(x, y, \theta) = (1, 1, 0)$. The convergence rate seems to decrease as the noise parameter is increased.

3.2 Autograder

Authored by Caden Moore

To ensure we had an accurate MCL model built up in simulation we required a computationally numerical way of determining if what our simulation does is accurate which is why we were given the autograder. The autograder consists of a staff solution for three simulation scenarios with noise, slight noise, and complete absence of noise. The staff solution is the best optimal solution we could strive for implementing a MCL method in simulation. For such a test to work we needed to compare how our MCL model worked in an identical scenario for our simulation versus the staff simulation which was done automatically by the autograder. The way both the staff's solution and our grade was determined was by calculating the solution's mean absolute deviation from the true ground trajectory and assigning a grade based upon that. Our solution was made with our particle filter and put to the test against the staff's solution using the staff's particle filter, and our autograded mean absolute deviation was compared to the staff's mean absolute deviation, and our grade depended on how much our deviation deviates from the staff's solution. Upon adjusting the number of particles to improve the speed at which our MCl simulation ran, our final grade for the autograder was very good with a rounded grade of 95 percent which ultimately meant that our solution compared similarly with the staff's mean absolute deviation. One thing we found was that we got a lower grade when it appeared our simulation solution matched better with the true ground trajectory and performed better than the staff's simulation solution and we had to actively try to make it worse for a better grade. One reason we think this may be is because the grade was calculated using $max(0, 1 - max(0, d_{submission} - d_{ref}))$. This would mean that we would be penalized for being further away from the staff solution rather than just from the true ground trajectory, so we would perform better than the staff solution when compared to true ground trajectory but the autograder did not account for the fact we were closer to true ground trajectory because we were off from the staff solution, not true ground trajectory. Thus we had to bog down our model slightly to achieve a higher grade to match the staff solution better.

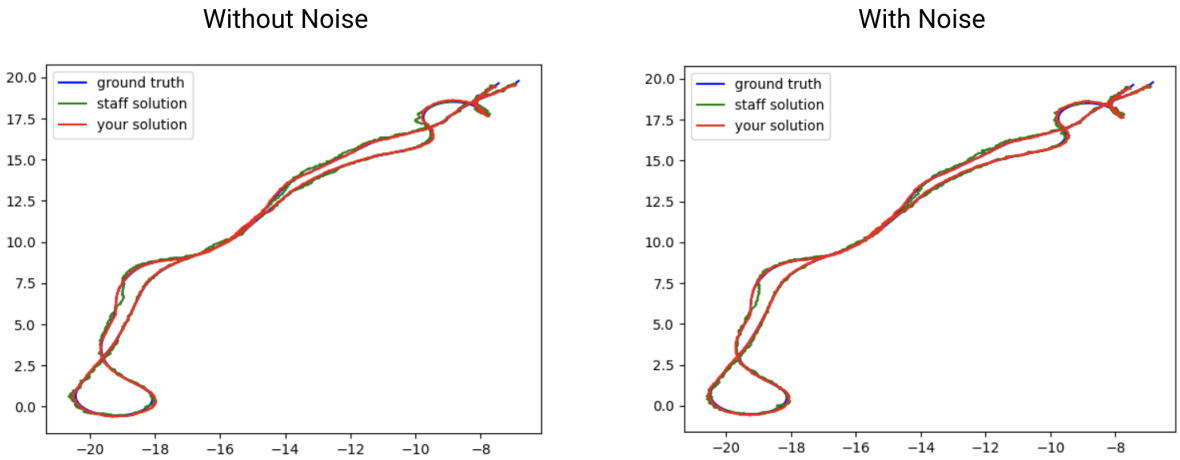

Figure 10:

3.3 Real World

Authored by Caden Moore

Our real world evaluation consisted of qualitatively analyzing how our localization model worked in RViz overlaid on the map and quantitatively measuring our convergence rate, spread of our particles at peak localization, and our cumulative error over time. We initially started out by identifying what we thought would be a relatively easy place to determine an initial pose for our robot. This was crucial because we needed to know that we would be starting in the same position for every real-world evaluation run. We chose a portion of the Stata basement right in front of the pillar in front of room 081 because we figured we could estimate the initial pose of the robot better in front of this pillar than anywhere else on the map. The pillar provided a great reference point not only for direction but also location. Upon initializing in front of this pillar we would manually operate the robot using the controller and try to drive in a chaotic and erratic manner to screw up our localization method as best as we could, which meant we would serpentine through the hallways getting close to the walls, going in circles and going in reverse; in essence, testing the robustness of our implementation. While the test was underway we would

qualitatively analyze how well the localization model was working on our robot by seeing how well the LIDAR scan data overlaid on the map while we were driving, and how the LIDAR measurements line up with perceived simulated walls. During every trial that we performed, the LIDAR scan data matched almost perfectly consistently, meaning that the robot was accurately sensing its environment.

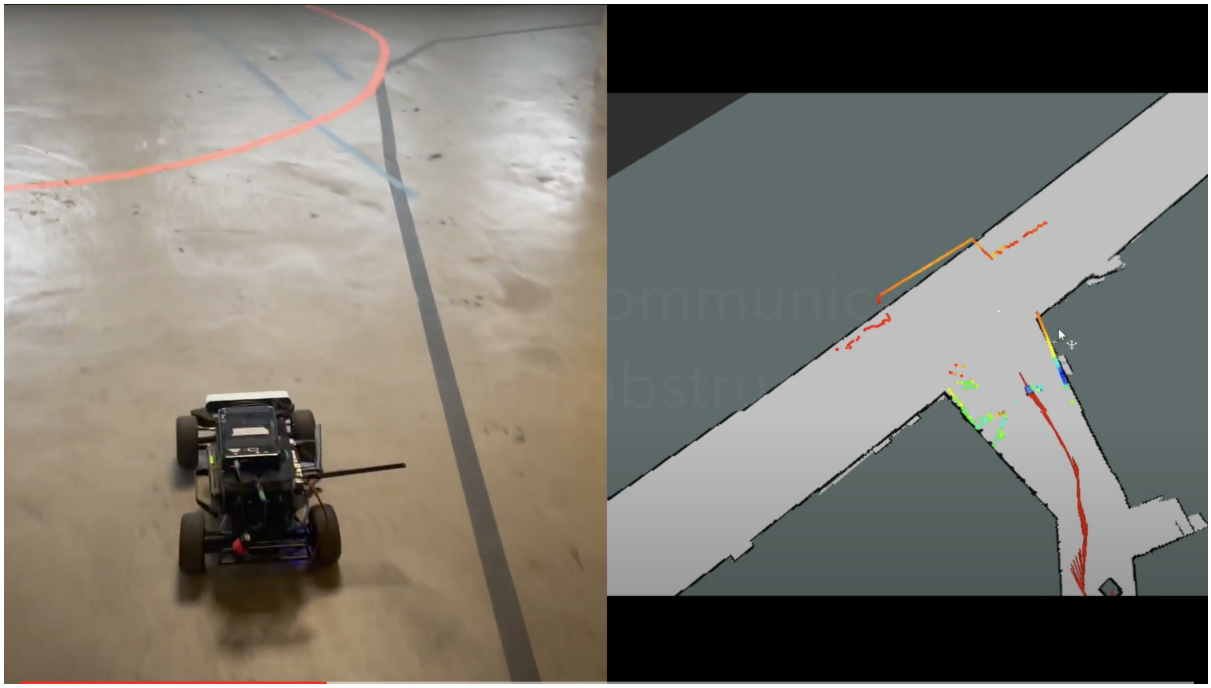

Figure 11:

There were no times when the LIDAR scan overlay was so visibly far off that it was not able to be compared to the map underneath, but more importantly the LIDAR scan overlay never got far off at all. In addition to this, we displayed a slime trail of sorts that displayed the predicted pose of the robot at each current time step at a rate of 10 hertz and the predicted pose of the robot was never substantially off, they were almost always spot on. Upon analyzing the slime trail afterward it was fairly accurate to where we had driven in the trials. In addition to this, the point cloud we generated was always centered on the robot. The particle filter was fairly responsive and was robust to rapid movements in odometry in our trials. This signified to us that our particle filter works well and reliably.

To quantitatively evaluate our model in the real world, we analyzed the spread of the particles when converged on the robot, and we analyzed the convergence rate of the particles. To calculate the spread of the particles, we took the max particle x and y coordinates and subtracted the minimum particle x and y coordinates. This would give us an idea on how far apart the furthest particles were from one another. Once this value remained consistently small we determined that this was our spread for our particles. The spread for our robot is roughly around 20 cm on average which is reasonably good. We came to this determination because it is on the same order of magnitude as the size of the robot itself, which is good for the real world. To determine convergence rate we found the time that it took the particles to go from max spread to the minimum spread that we found before. We did this by publishing and recording rostime and the spread in a rosbag. We analyzed the data from the rosbag and found that the average convergence rate for the particles is .034 seconds which is reasonably fast. The time it takes to converge is minuscule meaning that our model works well. How we gathered quantitative data of our accuracy in the real world was by measuring our cumulative error over time. We achieved this by ending our trial run directly in front of a wall, measuring the distance of the robot from said wall in simulation, measuring the distance of the robot to the wall in the real world, and comparing the two values. Using this method, we never achieved error greater than .22 meters. Overall, using both qualitative and quantitative measurement metrics we determined that our particle filter and MCL model

work sufficiently.

4 Conclusion

Authored by Vittal Thirumalai, edited by Caden Moore

In this lab, we implemented a fully functioning MCL algorithm to allow accurate localization of our racecar in a simulated environment. We then deployed and tuned our algorithm on the actual car, allowing the robot to reliably locate and track itself within the basement of STATA. This exactly achieved the goals we had when coming into our lab.

During the process, we encountered numerous obstacles and bugs as is expected with the combination of hardware and software, but we were able to handle this more effectively having completed a couple of labs previously and gaining more experience. Furthermore, by handling noise explicitly in the simulation, the transition to the real car was much smoother and required a lot less debugging. This emphasized to us the value of simulations and accounting for noise.

Though our MCL implementation works reasonably well, there are definitely ways to improve and optimize it given more time. For example, we could more finely tune the noise parameter to optimize it based on the map in consideration. We could also experiment with other methods of particle initialization than just the normal Gaussian distribution.

In addition to MCL localization, we also experimented with SLAM through Google Cartographer and learned how SLAM can be used in simulation to dynamically localize and create a map. We were able to successfully use Cartographer and visualize the created map in RViz. The task of dynamic localization is a crucial aspect of robot control since a robot needs to know where it is in order to allow higher-level path planning, which we will tackle in the next lab.

5 Lessons Learned

5.1 Henry Heiberger

Covering content that I knew almost nothing about before taking the class, this lab provided numerous learning opportunities for me.

Regarding technical lessons, as one of the most theory-heavy labs so far, this required me to perform an extensive amount of research online before I could even begin implementing the MCL algorithm. Doing this helped increase my skills at quickly skimming advanced research papers and textbooks in order to find the information needed to make progress on the algorithm. Furthermore, this lab was very unique in that we needed to actually add noise into our motion model in order for it to perform well outside of simulation, not remove it. This showed me the importance of thinking about how to create models that not only perform well in an ideal simulation environment but also perform well in execution. Finally, through the process of transferring our localization algorithm to the real robot, we ran into many hardware and ROS issues that delayed our deployment. This vastly increased my familiarity with the robot and ROS debugging skills, something that will hopefully be useful in the future.

Regarding communications lessons, the technical complexity of this lab really required me to think of ways in order to present the information in our briefing and report so that it was understandable to a diverse audience. This is a skill that takes practice, and I am glad that I am able to spend a fair bit of time on it in this class. Beyond this, I feel like the briefing presentation for this lab really emphasized the importance of practice, where being able to run through the slides a few times together before we presented really made it seem like everyone in our group was much more relaxed and well-spoken.

5.2 Benjamin Soria

I worked extensively with testing, data analysis, and capturing evidence of our racecar working. Testing required time set aside for multiple runs, as well as coordinating with my teammates when I couldn't capture data on my own. Luckily, implementing noise at several stages in our simulated environment resulted in a smooth transition into the real environment. However, we ran into obstacles on the first day of data collection, and the issues were not resolved until the Tuesday before our briefing. Our controller would not communicate with our racecar. I learned valuable debugging skills, as I tried to fix this issue. I reset my computer, tried communicating through a teammate's computer, restarted the racecar and allowed the capacitors to discharge, used another group's controller, and tried to echo the controller commands. Finally, we realized the issue was with the USB tail. We plugged the controller's USB directly into the racecar's dock and all of a sudden we had communication established.
Another big takeaway from this lab is that incorporating noise into our implementation while testing in simulation resulted in a smooth transition into the real world. For several labs, we had been trying to minimize noise and deal with uncertainty in our LIDAR data. In this lab, we purposefully modeled the noise we could expect to see in the real world while still developing in simulation. This resulted, quite surprisingly, in enabling us to observe a working localization implementation on our first couple rounds of testing in the real world.

5.3 Caden Moore

I worked on implementing our model and analyzing data on our robot in the real world and I assisted in code development and algorithm logic along the way. Although I never had an entire section dedicated to me I tried to help whenever I could and helped in a lot of the sections/modules, speeding up the process along the way. I also helped in gathering data and experimental evaluation for the model. I learned valuable communication skills this lab because I had to code in conjunction with others using only one computer, not multiple (especially when getting the experimental data, it was easier to just use one computer). As such, I had to communicate how I would code what I was thinking verbally much more effectively this lab. I also helped with lots of hardware and software issues that arose when implementing the model onto our robot and when doing experimental evaluation.

Monte Carlo Localization was a hard concept for me to grab at first. I did not know how it worked and where to start but ultimately I did figure it out in the end. I watched a couple of videos and read a paper and I think I have an alright understanding of the method now. When learning about this algorithm it made me realize that I would like to learn more about things like this, and that there are a lot of things in the world that I still do not know. I would like to pursue and gain more knowledge just like I did in this lab and I plan to do more research into the subject of localization.

5.4 Alazar Lemma

For technical lessons, MCL was a very difficult algorithm for me to wrap my head around, so I spent a massive amount of time trying to understand the method enough so that I could participate in the lab. I spent a lot of time rewatching lectures in class, and on YouTube, watching people explain the concepts to me in a way that I could understand. I learned how I like to learn exceptionally difficult topics, and how to effectively understand concepts more quickly. During the lab, we ran into a lot of issues with ROS and the hardware, so we practiced valuable debugging skills trying to debug the racecar.

For communication lessons, I mainly practiced my presentation skills, as I tend to struggle a lot with presenting. Learning how to deal with the nervousness was a key lesson for me for this lab.

5.5 Vittal Thirumalai

Considering the technical side of this lab, I learned a great deal about localization and mapping which I hadn't known much about before. I first spent a decent amount of time understanding how MCL works, and the initial theory side assignment was very helpful in doing so. By writing the particle initialization code, I learned about using Gaussian distributions and RViz visualization of pose arrays. In addition, I encountered many obstacles with installing and using Cartographer, so this process gave me further experience with debugging and RViz visualizations. Finally, as a group, we learned that accounting for

noise in the simulation is very effective at reducing most bugs when put onto the physical car. This highlights the value of proper testing in simulation before using the physical hardware on the robot.

Considering the communication side, I learned to better prepare and present in a confident manner. I also learned to better contribute dynamically during the QA section by stepping up when the TAs ask questions. Overall, I gained further experience working as a team, parallelizing and splitting up work, and communicating effectively.