Lab #5 Report: Implementing Monte Localization to Identify Vehicle Position in Enclosed Spaces

Team #14

Isabella Do Orozco
Spring Lin
Jace Lu
Kristoff Misquitta
Mark Razanau

RSS

April 10, 2024

1 Introduction (Kristoff, Spring)

Lab #5 addresses the challenge of robot localization, or how an agent determines its position and orientation given a map of the world. This is a significant step forward from prior work, where the robot responded to local geometric features, following walls and lines, but possessed no ability to reason about its place in the global environment. Here, localization is achieved via a computationally tractable form of Bayes filtering, known as particle filtering or Monte Carlo Localization (MCL). Such "particles" represent potential hypotheses about the state of the system being estimated.

In the context of the racecar, the state at hand is a possible pose (position and orientation defined by x, y, θ) of the robot within its environment. Initially, the particles are distributed across the map, representing various possible positions and orientations of the robot. As the robot moves and receives sensor measurements, the particles are updated based on a motion model and sensor model. Particles that are consistent with the observed sensor measurements are assigned higher weights, while particles inconsistent with the measurements receive lower weights. In a resampling step, particles with higher weights are more likely to be duplicated, while particles with lower weights are more likely to be removed. The motion model integrates the sampled particles forward in time based on velocity data. The cycle of prediction, measurement update, and resampling is repeated over time as the robot moves and receives new sensor data.

By maintaining a population of particles representing different hypotheses about the robot's pose and updating them based on motion and sensor information, the particle filter estimates the robot's position and orientation. The final estimated pose is typically computed as a weighted average of the particles' poses, ensuring robustness to uncertainty and noise.

Our specific tasks for this lab were to develop a motion model and sensor model, then integrate them in a functional particle filter. The development process began with odometry readings, LiDAR scans, and noise replicated in simulation. The program was then deployed on the robot and its localization performance evaluated in the Stata Center basement. Our work validates MCL for use in robust, location-aware autonomous navigation. More immediately, it will be an asset for accurate path plan execution in Lab #6.

2 Technical Approach

Our localization algorithm is separated into three core modules: motion model, sensor model, and particle filter.

2.1 Motion Model (Jace)

The first module of our localization algorithm, the motion model, is used to calculate the new positions of particles given the previous positions and the odometry reading.

2.1.1 Steps

Our motion model is split into three key steps. First, we retrieve the current positions of the particles x_{k-1} and the current odometry reading u_k.

$$x_{k-1} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} \tag{1}$$

$$u_k = \begin{bmatrix} \Delta x_k \\ \Delta y_k \\ \Delta \theta_k \end{bmatrix} \tag{2}$$

The second step is transforming the odometry reading from the body frame to the world frame. The positions of the particles are given in the world frame initially, but the odometry reading is given in the robot body frame. In order to correctly update the positions of the particles, we multiply the odometry reading by rotation matrix R_z to move from world frame to body frame.

$$R_z = \begin{bmatrix} \cos\theta_{k-1} & -\sin\theta_{k-1} & 0 \\ \sin\theta_{k-1} & \cos\theta_{k-1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The third and final step of the motion model is adding noise to the odometry reading. We use a normal distribution centered at zero with a standard deviation of 0.15 to generate the noise vector ϵ_k.

$$\epsilon_k = \begin{bmatrix} a_k \\ b_k \\ c_k \end{bmatrix} \tag{4}$$

After these three steps are performed, we sum the position vector x_{k-1}, the odometry vector $R_z u_k$, and the noise vector ϵ_k to obtain the new position vector x_k.

$$x_k = f(x_{k-1}, R_z u_k, \epsilon_k) \tag{5}$$

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \cos\theta_{k-1} & -\sin\theta_{k-1} & 0 \\ \sin\theta_{k-1} & \cos\theta_{k-1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta y_k \\ \Delta\theta_k \end{bmatrix} + \begin{bmatrix} a_k \\ b_k \\ c_k \end{bmatrix} \tag{6}$$

We then use the above equation to calculate the new positions of the particles every time a new odometry reading is received.

2.2 Sensor Model (Mark)

With the motion model complete, the next module in the system is the sensor model, which allows our robot to determine its real world position by comparing simulated LiDAR scan measurements with respect to its estimated pose.

2.2.1 Computing Scan Probability Values

In order to determine the probability that a measured distance z_k matches the given ground truth distance d, we can compute it as the sum of four key probabilities:

1) p_{hit}: the probability that the scan detects a known obstacle in the map such as a wall in the environment

2) p_{short}: the probability that the scan detects a shorter measurement in the environment, such a sudden obstacle

3) p_{max}: the probability that the scan returned is very large, likely from a missed measurement that is not returned to the sensor.

4) p_{rand}: the probability that a random measurement is returned due to an

unlikely event.

Additionally, we multiply each probability by a weighted term, defined respectively by α_{hit}, α_{short}, α_{max}, α_{rand}. This will give us the following formula:

$$\begin{aligned} p(z_k^{(i)}|x_k, m) =& \alpha_{hit} * p_{hit}(z_k^{(i)}|x_k, m) \\ &+ \alpha_{short} * p_{short}(z_k^{(i)}|x_k, m) \\ &+ \alpha_{max} * p_{max}(z_k^{(i)}|x_k, m) \\ &+ \alpha_{rand} * p_{rand}(z_k^{(i)}|x_k, m) \end{aligned} \tag{7}$$

However, as these probabilities represent a continuous set of values, it is crucial to avoid computing these probabilities repeatedly as it will significantly affect the performance of our localization method. As a result, we leverage an x by x look-up table that discretizes the scan values, computes the approximate probabilities, and stores them to be quickly retrieved during real-time evaluation. For our sensor model, we set x to 200. With this type of discretization method, we can write each of these probability cases as the following:

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

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

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

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

As the alpha weights must add up to 1, we use $\alpha_{hit} = 0.74$, $\alpha_{short} = 0.07$, $\alpha_{max} = 0.07$, and $\alpha_{rand} = 0.12$ with $\sigma = 8$. When considering the normalization of these probabilities within a look-up table of dimension x by x where each row is a given z_k value and each column represents a given d value, we pre-compute the respective $p(z_k^{(i)}|x_k, m)$ in the following way:

```
def precompute_sensor_model():
    for z_k in range(x):
        for d in range(x):
            compute p_hit
            add p_hit to table index [z_k][d]

    normalize p_hit values across columns
```

```
multiply table by alpha_hit value

for z_k in range(x):
    for d in range(x):
        compute p_short
        compute p_max
        compute p_rand
        multiply each p value by the respective alpha value
        add each weighted p value to table index [z_k][d]

normalize p values across columns
```

2.2.2 Evaluating The Probability of a Given Scan

Following the precomputation of probabilities between estimated and ground truth distances, we must then determine how the given set of LiDAR scans compares to the real world scan. Given a set of poses for simulated particles, we can take advantage of ray tracing to determine the scan that would result in the given position and orientation. For each simulated scan, we can then determine the probability of the measured distance from the ground truth distance by converting the values to integers and retrieving them from the table. To do so, each of these probabilities is multiplied over the range of the entire LiDAR scan, giving us the following probability equation for a given simulated scan:

$$p(z_k|x_k, m) = \prod_{i=1}^{n} p(z_k^{(i)}|x_k, m) \tag{8}$$

As the evaluation is dependent on calculating between large sets of scans, it is critical to optimize the methods to compute these probabilities. We utilize numpy methods such as np.prod() to efficiently multiply these probabilities with respect to the simulated scan and observation values. We also normalize this set of probabilities to ensure that they add up to 1. This will return to us a set of probabilities representing the likelihood that the given pose is true to the real world pose, giving us a better indication of where our robot is truly located in an environment.

2.3 Particle Filter (Isabella)

The Particle Filter utilizes the Motion and Sensor models to implement the MCL algorithm to localize the robot. As information from each module comes in, it keeps an updated list of "likely" particles. It does so with three main components

2.3.1 Particle Initialization

Based on a given initial pose (x, y, theta) of the robot, a normal distribution centered at this pose with a standard deviation of 0.1 is taken to create a set of size specified by the parameter $num_particles$. It is only when these particles have been initialized that the other components are allowed to trigger.

2.3.2 Odometry Updates

Whenever odometry data is received from the robot, the motion model results are used to update the particle positions. First, the twist from the robot is used to calculate Δx which is passed to be evaluated by Motion Model. The Particle Filter then uses the returned calculated states to update the list of particles.

2.3.3 Sensor Updates

Whenever sensor data is received from the robot, the LIDAR data is downsampled to 100 and then passed to the sensor model. The more LIDAR data that is passed, the more accurate the produced probabilities may be; however, this is also computationally intensive which can slow down processing. With a probability returned for each particle, they are randomly resampled with replacement using these weights. This causes the cloud of particles to condense down only to the most likely states.

2.3.4 Publishing

Anytime the list of particles is updated, the Particle Filter calculates the average pose of all the particles, publishing both this and the list of particles each to a separate topic. The odometry updates occur at a faster rate than the sensor updates, but the use of locks ensures both components are not attempting to update the particles at the same time. Working in tandem through the Particle Filter, the Motion Model and Sensor Model each have their place in evaluating probable locations for the robot. As the sensor model condenses the number of particles, the motion model adds variation and makes quicker updates to keep the localization moving.

3 Experimental Evaluation

3.1 Simulated Noise Evaluation (Kristoff)

The form and parameters of noise added to the model must be carefully considered to ensure the simulation is a faithful representation of on-car performance. Our solution inserts noise at two sites: pose initialization and the motion model update step. Here, we describe the quantitative process used for understanding the effect of motion model noise on the overall particle filter. A similar procedure was followed to qualitatively assess initialization noise.

3.1.1 Selection of Probability Distribution

A variety of candidate distributions for representing noise in the motion model were considered: normal, uniform, triangular, lognormal, and Poisson. The normal distribution was selected due to its ease of use with our coding toolchain (NumPy) and natural correspondence to the phenomenon at hand: a deterministic motion model update should be mostly correct, save for random spread around the center.

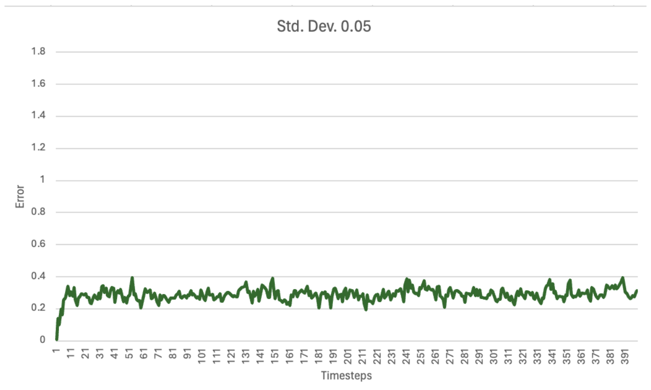
3.1.2 Results

The use of a normal distribution grants two degrees of testing freedom: varying the mean or varying the standard deviation. However, as it makes the most physical sense for the added noise to be centered at zero (not skewed one way or another), only the standard deviation was varied, at three levels: 0.05, 0.15, and 0.30. Consult Section 2.1.1 for a mathematical walkthrough of how this noise is added.

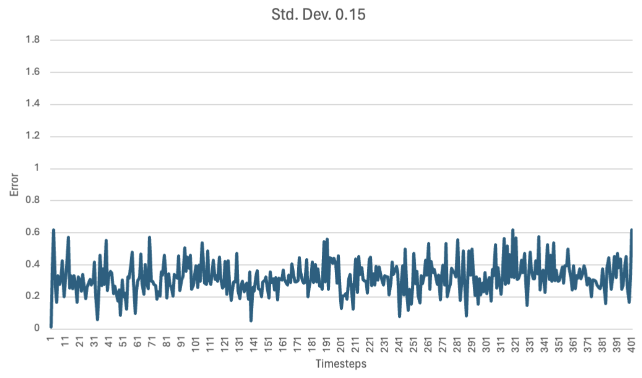
The distance between the robot's (noiseless) position and that computed after Gaussian distortion in the sensor model is used as the error metric, as described in Equation 9.

$$\sqrt{(x_{true} - x_{noisy})^2 + (y_{true} - y_{noisy})^2} \tag{9}$$

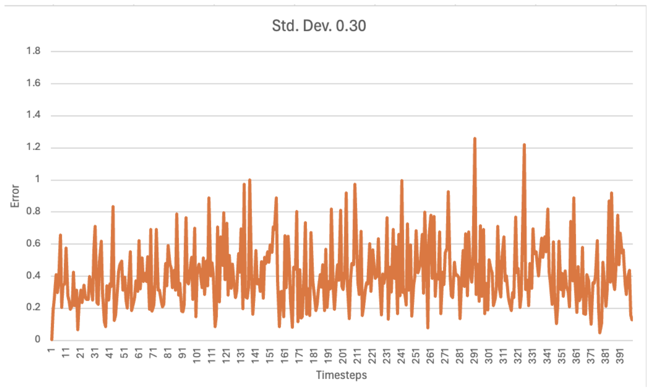
The robot is stationary in simulation for the duration of the test. The results are plotted in Figs. 1a-1c, with further summary metrics following in Table 1.



(a) Localization error under noise $N(0, 0.05)$.



(b) Localization error under noise $N(0, 0.15)$.



(c) Localization error under noise $N(0, 0.30)$.

Figure 1: Graphs of localization error under conditions of simulated noise with 3 standard deviations. As the standard deviation increases, the average and worst case errors both increase.

Table 1: Average and worst-case errors for varying Gaussian noise.

	$N(0, 0.05)$	$N(0, 0.15)$	$N(0, 0.30)$
Average error	0.289	0.319	0.409*
Worst-case error	0.384	0.618	1.260*

*Average and worst case errors are highest for the highest standard deviation case.

3.1.3 Analysis

Notice the expected superior performance of localization at the lowest standard deviation (average error of 0.289 for $N(0, 0.05)$) and the increasingly erratic - but still relatively accurate - behavior as the noise is amplified (relatively high worst-case error of 1.260 for $N(0.0.30)$). This indicates a functioning particle filter that should scale well with a range of real-life conditions.

3.2 Real World Evaluation (Jace)

The most important reason that a real world evaluation is needed for a localization algorithm is the essential role that it plays in safety-critical applications. For instance, the deployment of autonomous vehicles and the execution of robotic surgeries hinge on the reliability of localization to prevent accidents, injuries, and various safety hazards. When localization precision falters, the ensuing errors can have dire consequences. Moreover, localization serves as a crucial cog within larger systems that encompass mapping, navigation, and device interaction. The accuracy of localization is integral to the harmonious function of these systems, ensuring that errors are not compounded when these technologies operate in concert. The repercussions of inaccuracy are not localized; they can escalate as the system's reach extends. In addition, the insights gleaned from the assessment of localization models are a boon to research and development efforts. They afford us a granular understanding of real-world performance and limitations, which is instrumental in spurring the refinement and innovation of localization technologies. Such evaluations are not merely diagnostic but serve as a springboard for technological advancements.

3.2.1 Method

To ascertain the precision of the localization algorithm, we conducted an empirical analysis by comparing the vehicle's estimated position, as from the algorithm, against actual measurements obtained in a controlled environment. This process involved the utilization of a taped path within the confines of the Stata basement. The path was then meticulously delineated into eight discrete segments to facilitate detailed assessment. As the racecar navigated each segment, it was momentarily halted at the designated increments to allow for the recording of the algorithm's positional estimates. These data points, corresponding to the x and y coordinates, were then systematically retrieved by echoing the topic the estimated position is published to.

3.2.2 Results

We recorded the actual x and y coordinates of the path used to test the localization algorithm and compared them to the algorithm's predicted coordinates. The table and figures below illustrate this comparison between the two datasets.

Table 2: xy coordinates of the racecar.

Measured y (m)	Predicted y (m)	Measured x (m)	Predicted x (m)
0	0	0	0
0	-0.1321	1.82	1.4863
0	0.1719	3.64	3.2967
0	-0.2803	5.46	5.2859
0.5029	0.2323	5.9629	5.8366
3.2546	3.1486	5.9629	6.3423
6.0063	5.7806	5.9629	5.6835
8.758	8.4508	5.9629	6.2565

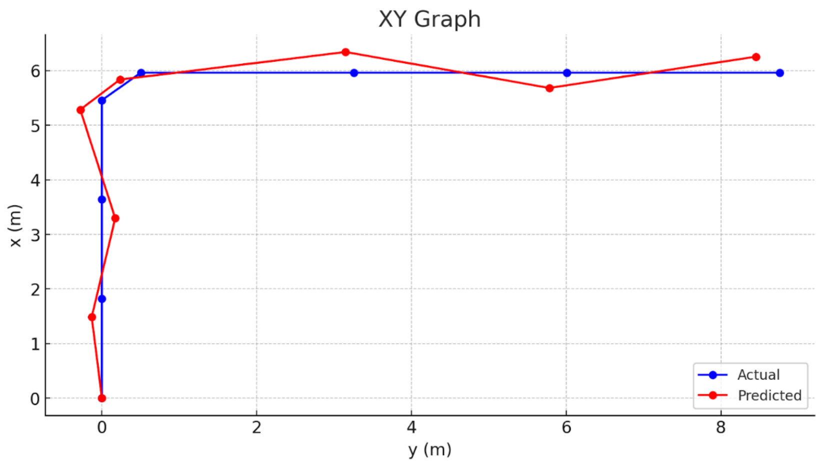


Figure 2: Paths taken by the racecar.

Table 3: xy errors of the predicted coordinates of the racecar.

y error (m)	x error (m)
0*	0*
-0.1321	-0.3337
0.1719	-0.3433
-0.2803	-0.1741
-0.2706	-0.1263
-0.106	0.3794
-0.2257	-0.2794
-0.3072	0.2936
Average y error: 16.08%	Average x error: 7.01%

*These error values were omitted in error calculations due to racecar being initialized on y=0 and x=0.

3.2.3 Analysis

As depicted in Figure 2 and Table 3, the localization algorithm attained an accuracy of 83.92% along the y-axis and 92.99% along the x-axis. The computation of error was conducted utilizing the subsequent equation:

$$\%error = \left| \frac{error}{measured} \right| \times 100 \tag{10}$$

The current level of accuracy achieved is commendable; however, it falls short of the precision required for high-stakes applications. With the present model, a racecar traversing a distance of 100 meters could experience a deviation of as much as 16.08 meters in the y-direction and 7.01 meters in the x-direction. Such discrepancies are significant and could severely impact the racecar's performance and safety.

3.3 Discussion(Spring)

At present, our implementation of the MCL algorithm for localizing the robot within its environment has proven successful. This achievement stems from the integration of two critical modules: the motion model and the sensor model. The motion model accurately predicts the robot's future state based on its current state and received control inputs, incorporating Gaussian noise from a normal distribution to simulate real-world conditions. Simultaneously, the sensor model governs how the robot's sensor readings influence the probability distribution of its estimated pose. Through precomputation and evaluation, these models handle the majority of computations related to the probabilities between actual and hypothesized LIDAR distances. We've optimized sensor computations by leveraging techniques such as normalization, discretization, and lookup tables, aided by efficient libraries like NumPy and vectorization for faster simulations.

The fusion of these modules occurs within the particle filter, where locks ensure synchronized utilization of both motion and sensor models to determine an "average" pose. Additionally, we introduce a second layer of Gaussian noise with a standard deviation of 0.15 within the particle filter, enhancing robustness to real-world noise. Simulation tests confirm the accurate localization of particles alongside the robot, while real-world experiments exhibit close accuracy in localization.

4 Conclusion (Spring)

Our current design of the MCL algorithm is based off of two modules: motion model and sensor model being married together in particle filter. We are able to obtain particles from LiDAR scans, which are updated with the help of motion model and sensor model. Particles are then assigned corresponding weights based on the consistency between predicted and real life measurements. In a resampling step, particles with higher weights are more likely to be duplicated, while particles with lower weights are more likely to be removed. The particle filter then utilizes the continuously updated particles to localize the robot by calculating the average pose of the particles. Additionally, odometry information is updated iteratively, though not simultaneously with particle localization.
This approach allows for robust localization within diverse environments, with careful consideration allotted to updating odometry, assigning proper weights to positions, and properly publishing odometry and average pose.

Moving forward, our focus shifts to further optimizing the localization algorithm, particularly concerning the evaluate method in sensor model. We plan to enhance code efficiency by leveraging NumPy's built-in functions and maximizing vectorization to eliminate current for loops. Moreover, we aim to streamline the number of particles and laser scan data processed to expedite processing time. In addition to the optimization efforts outlined, we will conduct further testing to thoroughly validate the performance of our localization system. Initially, we plan to utilize ROS bags in simulation environments to simulate various scenarios and stress-test our algorithm under different conditions. This step will allow us to analyze the system's behavior and fine-tune parameters before deploying it in real-world settings.

Once satisfied with the simulation results, we will transition to real-world experiments, where we will deploy the algorithm on our robot and collect data in diverse environments. By comparing the localization results obtained in simulation with those from real-world scenarios, we can assess the algorithm's robustness and accuracy across different conditions.

By adhering to Bayesian principles and employing probabilistic reasoning, MCL provides a robust solution to address uncertainties inherent in real-world robotic applications. This capability empowers the robot to navigate with confidence and precision across diverse environments, laying the groundwork for enhanced autonomy and performance.

5 Lessons Learned

5.1 Isabella

Throughout this lab, I learned more about "expecting the unexpected". Many times throughout working, I realized I needed to adjust my expectations for things working on a given schedule. For example, we had our code working at a great point, but every time after we seemed to encounter problems that led us to backtrack. I think accounting for the fact we are going to have many unexpected issues at the beginning will lead to less disappointment and help us keep a realistic view of how the lab may progress. Another important thing I would like to take forward with me is to keep the end goal in mind whenever working. For example, if we keep in mind that the robot necessitates non-computationally heavy work from the beginning, then we can attempt to start off in an easier place to attain this.

5.2 Spring

Throughout this class, especially lab #5, I have had the privilege to view robotics on a more up and personal level. Throughout the past week, I have had a concussion but with the help of my team, I was able to catch up almost seamlessly. Being able to better understand the relationship between motion model and sensor model, through two separate modules being married together with the particle filter. Seeing and practicing the transitions between world and body frame and optimizing with vectorization. Truly seeing these changes being implemented on the robot was very satisfying when it worked. However, as always, it was a struggle to do so. The overall experience was enlightening to say the least and we will take the lessons learned from this lab into lab #6.

5.3 Jace

Throughout the duration of Lab #5, I have come to recognize that my enthusiasm for robotics is not as fervent as I initially anticipated. Furthermore, I have acquired a deeper understanding of the efficiency benefits provided by numpy arrays over the utilization of for loops in the processing of substantial data sets. The theoretical knowledge gained from comprehending the interplay between the motion model, sensor model, and particle filter within a robust localization algorithm was intellectually stimulating. Nonetheless, the transition from code simulation to actual robotic implementation presented numerous hardware challenges that proved to be rather vexing. The disparity between simulation success and the multitude of issues faced when deploying code on the robot was stark, leading to extensive periods dedicated to troubleshooting. Indeed, debugging and rectifications consumed the majority of our time.

The overall experience was made more tolerable by the relentless determination and camaraderie of my team members. Their collective efforts—ranging from Mark's initiative to procure food from Masseeh during our late-night sessions to Spring's dedication despite her concussion—were instrumental in navigating the hurdles we faced. Their unwavering commitment was a crucial element in the successful culmination of our project.

5.4 Kristoff

I am excited by our use of advanced, mathematical theories in real engineering systems. Robustness to noise is often talked about with complementary algorithms, but the opportunity to see it in practice is rare. I'm eager to find more techniques in the domain of probabilistic robotics to test-drive on simpler systems.

I've also begun to appreciate the difficulty of building a cohesive product when it's comprised of large parts I did not own. In a CI setting, this requires active communication with teammates and often the use of visual aids, like flowcharts, to examine how ideas flow across the system. As the projects I tackle scale in complexity, I aim to place more trust in my partners and communication tools, so I can focus where my own areas of expertise lie.

5.5 Mark

Over the course of this class, I believe that I've begun to develop a much deeper understanding of how robotic systems and autonomous vehicles operate in the world around me. Wrapping up Lab 5, I can truly appreciate the complexities that underlie seemingly simple yet critical capabilities such as being able to localize oneself in an environment. By being able to implement these methods and algorithms, my understanding of these systems is much higher than if I had learned of these topics from a theoretical level. Not only does it allow me to directly see the effects of my code with respect to our robot's behavior, but it also allows me to understand the struggles that often comes with taking theory and applying it to a real world environment. With so much more variance and opportunity for elements to break, I can truly appreciate the difficulties that comes with implementing these systems that I often took for granted. A significant amount of testing and debugging goes into every one of these systems, and if our late night work sessions are indicative of how tedious this process can be, I can only respect the engineers that work behind the scenes to bring these autonomous robots to life.

Of course putting together such a complicated system is near impossible by yourself. This class has really stressed the importance of leveraging teammates to take responsibility of various components of these assignments, yet educate one another on the intricacies of their modules to put together a robust system. While this may seemingly come with the sacrifice of an in-depth understanding the system as a whole, having each teammate turn their focus on a specific module is the only way for our team to be able to achieve these tight deadlines. As a result, I have learned to put a lot of trust in my teammates to not only be able to complete their tasks, but also have the patience to help me understand how their systems work. This has also made me more attentive of my responsibility within the team, and holding myself accountable to ensure that I am not slowing down the work of my teammates. Ultimately, I truly appreciate the stubbornness and kindness of each and every one of my teammates. This allows for an amazing work environment where, even when our robot may be failing miserably, we can still find ways to enjoy the time working on the lab together.