Lab 5 Report: Localization

Team 12

Adan Abu Naaj
Amina Luvsanchultem
Andy Yu
Arthur Hu
Dylan Gaillard

6.4200: Robotics Science and Systems

April 12, 2025

1 Introduction

Author(s): Amina Luvsanchultem

In this lab, we implemented a Monte Carlo Localization system to enable the robot to pinpoint its position and orientation using odometry and LiDAR scan data. This functionality provides a fundamental basis for the robot's autonomous navigation around the map and builds on the control and sensing systems we developed in prior labs. In previous labs, we focused on the path and obstacles directly in front of the robot; however, in this lab, we shifted to a broader perspective of the robot's movement and position, allowing us to understand the robot's entire environment rather than a small segment. To ensure that our robot maintains an accurate estimate of its position and orientation on the map, we implemented three components: a sensor model, motion model, and particle filter.

Our solution focused on using a probabilistic approach to localization where we defined a set of weighted particles on the map, which represented estimates of the possible locations and states of the robot. The weights of each particle convey how well each particle aligns with the LiDAR sensor measurements — a higher weight

would mean the robot aligns well with the measured sensor readings and vice versa. These weights are determined by the sensor model, which compares predicted LiDAR measurements with the actual ones using a precomputed probability distribution table to assign a weight to each particle. The motion model updates the robot's position and orientation based on provided odometry data with Gaussian noise. The particle filter combines the data provided by the sensor model and motion model to update and resample the predictions of the robot's position, ensuring that we have a diverse and accurate range of predictions.

2 Technical Approach

Author(s): Amina Luvsanchultem, Andy Yu

2.1 Problem Statement

The goal of this lab was to implement a Monte Carlo Localization (MCL) algorithm in ROS2 Humble that, given a reasonable initial pose, could accurately estimate the racecar's position in the Stata basement using LiDAR data and a provided map of the basement. Our MCL implementation relied on two core components: the motion model, which updates particle positions based on the racecar's odometry or movement data, and the sensor model, which assigns probabilities to each particle based on how well its predicted LiDAR scan matches the observed scan—effectively measuring how likely each particle is to represent the car's true position.

2.2 Initial Setup

Our initial setup for this lab consisted of a racecar robot equipped with various sensors and:

1. an onboard LiDAR sensor
2. 4-wheel-drive and steering motors (an NiMH battery)
3. an NVIDIA Jetson Nanodeveloper kit (and battery) running ROS2 in a docker container
4. a router equipped with ethernet for communicating with the Jetson

5. a Logitech joystick controller for teleoperation

2.3 Particle Initialization

Our Monte Carlo Localization (MCL) implementation begins with a manually provided initial position. Once this pose is given, particle locations are initialized by sampling from a Gaussian distribution centered around it. Specifically, we apply a standard deviation of 0.5 for the X and Y coordinates, and 0.1 for the theta orientation. Each particle is assigned an equal initial weight, resulting in a uniform distribution of probabilities (1 / total number of particles). Only after this initialization step do the sensor and motion models begin updating the particles, adjusting their weights, and publishing the robot's estimated pose as part of the localization loop.

2.4 Sensor Model

The framework for our sensor model focuses on implementing a [probabilistic methodology] for determining the weights of each particle, where each particle on the map represents a possible position of the robot. The main objective of the sensor model is to determine how well the observed LiDAR sensor readings match with those of the robot's predicted positions. The sensor model first defines a precomputed table mapping the ground truth distance to a range of possible discretized LiDAR distance measurements — ours ranged from 0 to 200 meters as given. Each probability in the table represents the likelihood of measuring a distance measurement given the ground truth distance. The likelihoods are determined using four different distributions:

We determined our max distance, z_{max}, to be 200. $z_k^{(i)}$ is the measured range, and the ground truth range is d.

$p_{\text{hit}}(z_k^{(i)} \mid d)$ accounts for when the measured value is close to the ground truth value.

$$p_{\text{hit}}(z_k^{(i)} \mid d) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z_k^{(i)}-d)^2}{2\sigma^2}\right) & \text{if } 0 \leq z_k^{(i)} \leq z_{\max} \\ 0 & \text{otherwise} \end{cases}$$

$p_{\text{short}}(z_k^{(i)}$ accounts for when the measured values are shorter than expected.

$$p_{\text{short}}(z_k^{(i)} \mid d) = \frac{2}{d} \begin{cases} 1 - \frac{z_k^{(i)}}{d} & \text{if } 0 \leq z_k^{(i)} \leq z_{\text{max}} \text{and } d \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$p_{\text{max}}(z_k^{(i)} \mid d)$ accounts for when the sensor outputs the maximum distance reading, which usually means that there are no obstacles within the range of the LiDAR.

$$p_{\text{max}}(z_k^{(i)} \mid d) = \begin{cases} \frac{1}{\epsilon} & \text{if } z_{max} - \epsilon \leq z_k^{(i)} \leq z_{max} \\ 0 & \text{otherwise} \end{cases}$$

$p_{\text{rand}}(z_k^{(i)} \mid d)$ accounts for randomness in the readings.

$$p_{\text{rand}}(z_k^{(i)} \mid d) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_k^{(i)} \leq z_{max} \\ 0 & \text{otherwise} \end{cases}$$

To determine the likelihood of each pairing of ground truth and measured distance values in the precomputed sensor table, we would need to combine these four distributions using a weighted average where α_{hit}, α_{short}, α_{max}, α_{rand} are the associated weights. In our implementation, the alpha values were 0.74, 0.07, 0.07, and 0.12, respectively.

$$p(z_k^{(i)} \mid d) = \alpha_{\text{hit}} \cdot p_{\text{hit}}(z_k^{(i)} \mid d) + \alpha_{\text{short}} \cdot p_{\text{short}}(z_k^{(i)} \mid d) + \alpha_{\text{max}} \cdot p_{\text{max}}(z_k^{(i)} \mid d) + \alpha_{\text{rand}} \cdot p_{\text{rand}}(z_k^{(i)} \mid d)$$

We thus calculate this value for each pairing of ground truth and measured distance values. Then, to determine the particle weights, we used the scaled observed LiDAR data from the car and looked up the probability of each observation in the precomputed sensor table, given the ground truth sensor reading. We then compute the product of the probability of the observed data given the ground truth data across all sensor scans or beams to determine the likelihood of each particle — the particle weights. We can see this process described further in the pseudocode below.

```
def evaluate(self, particles, observation):
        scale and clip observation to contain distances up to 200
        get the ground truth scan data
        scale and clip the scans to contain distances up to 200

        initialize a probability array
        for scan in scans:
            new probability = the product of the probabilities
            at indices of scan and observation in the precomputed
            table

            append new probability to probability array

        return the probability array
```

2.5 Resampling

In our sensor model, we perform resampling to generate a new set of particles from the current array. This involves sampling particles with replacement according to their normalized probabilities. Each sampled particle carries over both its pose and associated probability. The result is a new particle set that concentrates more heavily around high-probability regions from the previous time step.

While this algorithm worked well in simulation—successfully localizing the robot on a map of the Stata basement—real-world testing revealed additional challenges. Specifically, particles would sometimes pass through or remain inside walls, and the robot struggled to localize in long, featureless hallways.

To address the wall issue, we added a check before finalizing the new particle poses and weights: if a particle is found to be inside a wall or an occupied space (determined by the occupant grid of the map), we set its weight to zero.

To handle hallway localization issues, we introduced a pre-resampling check to maintain particle diversity in ambiguous environments. We compute the effective sample size using the equation:

$$n_{eff} = \frac{1}{\sum p_i^2}$$

where p_i is the weight of particle i. In an ideal case with uniform weights, n_{eff} equals the total number of particles. As particles diverge and few maintain high probabilities, n_{eff} drops. If n_{eff} falls below 60% of the total particle count (a tunable threshold we kept between 50% and 70%), we trigger resampling to maintain a reasonably tight distribution around our predicted location. Otherwise, we retain the existing particles and weights, assuming the distribution is adequately spread.

2.6 Motion Model

The motion model updates particle positions using odometry or robot movement data. This odometry consists of translational and angular velocities in meters per second and radians per second, which we scale by the subscriber's read frequency to reflect motion over time accurately. Once we obtain the robot-relative velocities in X, Y, and theta, we add Gaussian noise to better simulate real-world uncertainty and ensure sufficient particle diversity for effective localization. This noise is optional and can be toggled off. We apply Gaussian noise to each particle's X, Y, and theta values, centered around their original values with tunable variances. We found that variances of 0.1, 0.1, and 0.05 for X, Y, and theta respectively worked well. The noisy motion data is then converted into a transformation matrix representing movement relative to the previous pose. We apply the same process to the previous particle poses with respect to the map. This allows us to easily multiply matrices and compute the new particle poses in the map frame. Finally, we convert these updated matrices back into [X, Y, theta] format, both to match the function docstring and test expectations and because it is more intuitive and memory-efficient for use in the rest of our pipeline.

2.7 Localization Loop and Pose Estimation

To summarize, our localization loop proceeds as follows (note that this is not a strictly chronological list—the sensor and motion models may run in either order depending on timing):

1. An initial position is provided, which initializes the particles and their weights.
2. The motion model updates particle positions based on odometry data from the racecar.
3. The sensor model adjusts particle weights based on the latest LiDAR scan.
4. A predicted pose of the racecar is computed from the weighted average of the particle set. This process is detailed further below.

After each call to our sensor or motion model, we publish a weighted average of the particles' X and Y positions, using their probabilities as weights. For theta, we compute a weighted circular mean to account for its wraparound nature. The resulting [X, T, theta] values represent the localization algorithm's best estimate of the car's position and orientation relative to the map.

3 Experimental Evaluation

Author(s): Dylan Gaillard, Andy Yu

3.1 Simulation Evaluation

To quantitatively evaluate the performance of the particle filter, we first tested it in simulation, which allowed for a direct quantitative comparison between our localization and ground truth positions on the map—which real-world testing does not offer. This consisted of analyzing three metrics on the localized (x, y) position of our car: cross-track error (position error perpendicular to the heading of the car), total position error, and convergence (how long it takes to localize the car in a stationary position from an initial distribution of particles).

The cross-track and position error metrics are defined mathematically as follows:

$$e_{\texttt{cross}} = (-\sin\theta_t)(x - x_t) + (\cos\theta_t)(y - y_t)$$

$$e_{\texttt{pos}} = \sqrt{(x - x_t)^2 + (y - y_t)^2}$$

where (x_t, y_t, θ_t) are the ground truth position and heading, and (x, y, θ) are the respective localization values. To analyze convergence, we also observe position error.

We tested both low- and high-noise scenarios by adjusting the $\sigma_{x,y}$ and σ_θ standard deviation parameters of the gaussian distribution from which noisy odometry values are drawn. These were $(\sigma_{x,y}, \sigma_\theta) = (0.1, 0.075)$ and $(0.3, 0.1)$, respectively. These tests consisted of localizing the car as it followed a preset path throughout the map. The preset path consists of a straight path down a hallway, followed by a right turn and second straight path, taking about 27 seconds total to navigate.

To test convergence, given a stationary car and initial position estimate/particles drawn from a gaussian distribution with parameters $(\sigma_{x,y}, \sigma_\theta) = (2.5, 0.5)$, we observed the time necessary for position error to reach a steady-state value.

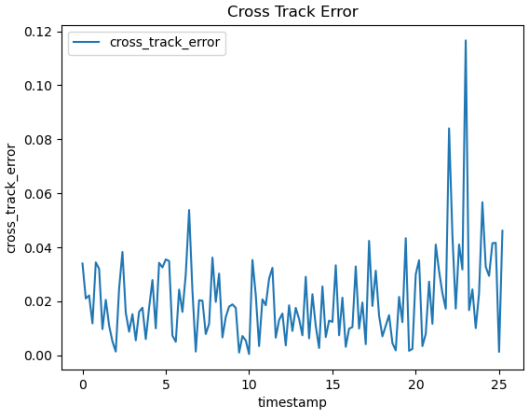

Figure 1: Cross track error (m) over time (s) as the car navigates the preset path with $(\sigma_{x,y}, \sigma_\theta) = (0.1, 0.075)$.

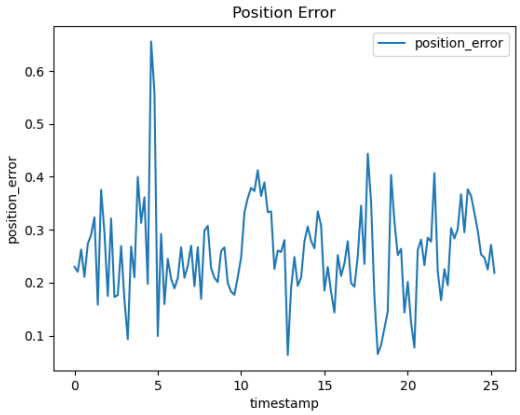

Figure 2: Position error (m) over time (s) as the car navigates the preset path with $(\sigma_{x,y}, \sigma_\theta) = (0.1, 0.075)$

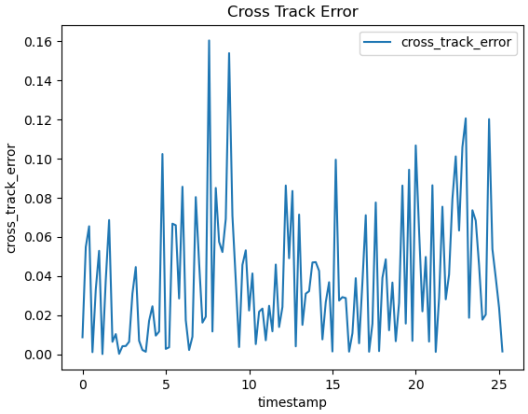

Figure 3: Cross track error (m) over time (s) as the car navigates the preset path with $(\sigma_{x,y}, \sigma_\theta) = (0.3, 0.1)$

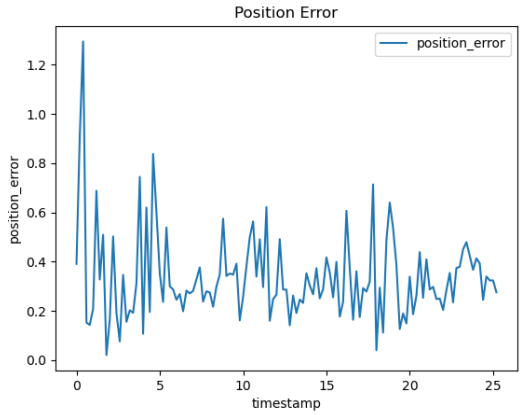

Figure 4: Position error (m) over time (s) as the car navigates the preset path with $(\sigma_{x,y}, \sigma_\theta) = (0.3, 0.1)$

As seen in Figures 1-4, note that both the cross-track and position error plots display signifiant noise, which is a resut of the motion model applying noise to particle odometry transforms, therefore increasing error, followed by particle filtering, localizing the car more accurately and decreasing error. Further, as expected in somewhat featureless hallways, the quality of localization deteriorates; this is visible in the higher position error at roughly 5, 10, and 17 seconds in both high and low-noise conditions. We also note that the spike

in initial position error in Figure 4 is due to initial conditions, and that the peak in final error in Figure 1 is possibly due to very fine-detailed areas of the map after the turn in the path (likely boxes in the real world), causing LiDAR scans and thus particle weights to be highly sensitive to changes in position. Since the number of particles in the simulation is limited, this causes the particle filter to be highly sensitive to cases where few particles are near the exact actual position of the car, increasing error.

Lastly, under low noise conditions, the average cross track and position error are roughly .02 and .26 meters. Under high noise conditions, we note that the average values are roughly .04 and 0.34. This is expected, as the higher motion model noise increases the uncertainty of localization. However, in both the low and high noise case the error is relatively compared to the size of the car, indicating robust localization.

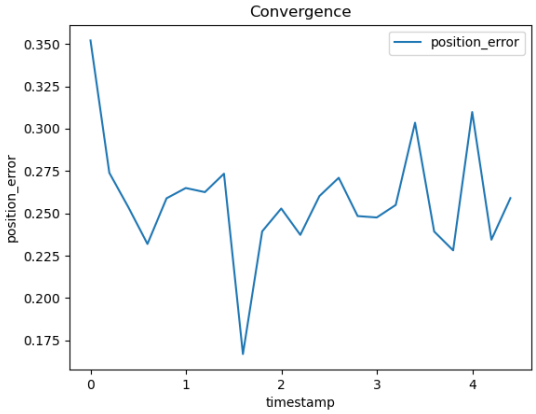

Figure 5: Convergence plot of stationary localization position error with particles sampled from a gaussian distribution with parameters $(\sigma_{x,y}, \sigma_\theta) = (2.5, 0.5)$, and motion model parameters $(\sigma_{x,y}, \sigma_\theta) = (0.1, 0.075)$. As time passes and the particle filter localizes the car more accurately, the position error decreases.

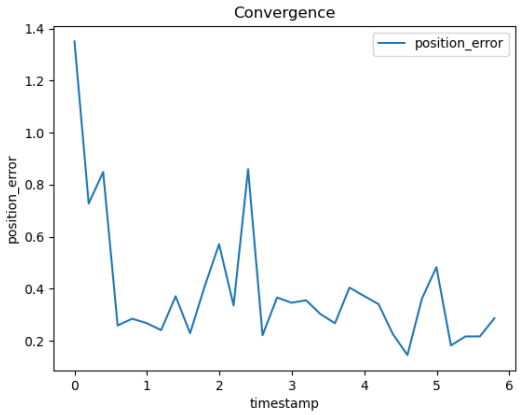

Figure 6: Convergence plot of stationary localization position error with particles sampled from a gaussian distribution with parameters $(\sigma_{x,y}, \sigma_\theta) = (2.5, 0.5)$, and motion model parameters $(\sigma_{x,y}, \sigma_\theta) = (0.3, 0.1)$. As time passes and the particle filter localizes the car more accurately, the position error decreases.

Regarding convergence error plots as seen in Figures 5 and 6, we note that our particle filter's error converges to its steady-state value of about 0.25 meters in less than 1 second in both the high- and low-noise cases, indicating fast and effective localization. Also note that the convergence speed is slightly faster in the low-noise case, as ex-

pected.

As a whole, our in-simulation results support the conclusion that our particle filter is both robust to low- and high-noise conditions, and localizes accurately.

3.2 Real World Evaluation

We used qualitative methods to evaluate our localization performance in the real world. Specifically, we assessed how well the projected laser scan data—based on the predicted pose—aligned with surrounding walls and obstacles. This evaluation was conducted by teleoperating the racecar around the large loop in the provided Stata basement map and visually inspecting how well the LiDAR points overlapped with the walls surrounding the predicted pose. To collect and visualize this data, we initialized the Monte Carlo localization with a manually set initial pose while connected to the router. We then recorded a ROSBAG during the run, which could later be retrieved and processed to analyze the racecar's estimated poses once the connection to the router was re-established.

The racecar performs well through most of the loop, particularly in the featureless hallways—where the map closely matches the real-world environment. However, issues arise in the top portion of the loop, where several boxes present in the real world are not reflected in the provided map. As a result, the LiDAR data frequently shows narrower corridors than those represented in the map, leading the localization algorithm to incorrectly favor a narrower section of the hallway. This misalignment is not corrected by the motion model, which causes the localization to fail.

Overall, our qualitative offshoots suggest that the localization algorithm can perform reliably in the Stata basement—provided the map accurately reflects the environment. When discrepancies exist, however, the system struggles to maintain an accurate estimate of the robot's position.

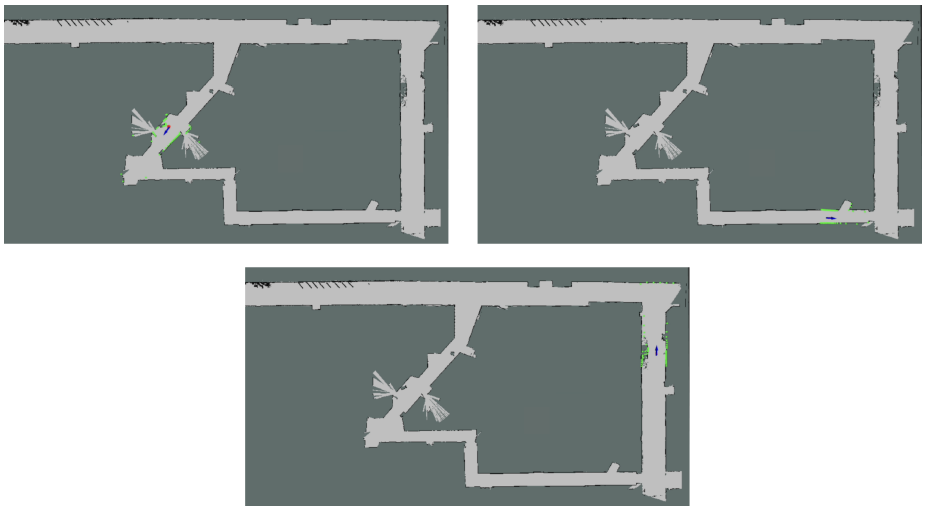

Figure 7: The localization performs well for three-quarters of the loop. The laser scan (green dots) projected on the position of the predicted pose (blue arrow) matches the surrounding walls (black) well.

Figure 8: The localization fails at the top of the loop. The laser scan (green dots) projected on the position of the predicted pose (blue arrow) does not match the surrounding walls (black) and never recovers.

A more rigorous quantitative evaluation would have involved computing a metric based on the proportion of LiDAR scan points—projected from the predicted pose—that landed on occupied cells in the map. Due to time constraints, we were unable to implement this approach.

4 Conclusion

Author(s): Amina Luvsanchultem

In this design phase, we implemented a Monte Carlo Localization system that allowed our car to localize its position and orientation in the map using the LiDAR sensor data and odometry data. Using the sensor model, motion model, and particle filter, our localization model proves to be robust against noise and accurately estimates the robot's pose in simulation and the real world. In simulation, we can conclude that the particle filter reliably converges to an accurate estimate of the robot's position in low and high noise conditions. The position and cross-track errors remain within acceptable bounds. In the real-world localization cases, we noticed greater variability in ambiguous situations, such as in long hallways with non-unique features, when compared to the simulation. However, the localization system was ultimately able to provide an accurate estimate of the robot's pose. An area of improvement we noted included handling the situations of external noise in our LiDAR readings, such as the case where boxes were stacked along the walls — a situation not accounted for in our ground truth LiDAR data.

This lab highlighted the difference between the more reactive autonomous navigation strategies we had implemented in our previous labs and a more generalized and informed understanding of autonomous navigation. Although we haven't used our localization model for autonomous navigation yet, we can already see the benefit of utilizing localization to more robustly navigate autonomously. By maintaining an accurate understanding of the robot's position in a known map, we can plan for more complex environments and implement path planning on the robot in the future.

5 Lessons Learned

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

5.1 Andy Yu

The biggest lesson I've learned, and continue to learn, is the importance of data collection, particularly the need to justify our methods and clearly explain our results. Being repeatedly pressed during briefings for unsatisfactory explanations behind data collection methods and trends is not a pleasing experience, especially after spending so much time just trying to get a robust, functional algorithm running in the real world. It's frustrating to hit a wall at the data collection stage when crunch time arrives. However, I'm learning to better balance my priorities and to recognize that, especially in the context of this class, thoughtful and well-documented data collection is just as important, if not more so, than a working racecar.

5.2 Dylan Gaillard

The main lesson that I am able to draw from this lab relates to the overall process of data collection and performance evaluation.. Given that I have primarily worked on our problem approaches and methods in the past, additionally contributing to the evaluation and data collection stages of this lab allowed me to better develop and understand thought processes that can be useful in intuiting and assessing the different behaviors of a solution to a problem. For example, I feel that I gained a greater appreciation of how using different evaluation metrics, etc. can have a significant impact on one's ability to understand the performance of, in this case, the particle filter and occasionally reveal issues/insights that would not have been uncovered otherwise. Further, I will also note that the briefing experience that we had and the many questions related to our group's data analysis have also motivated me to be more thorough and seek further insight in future data collection situations.

5.3 Amina Luvsanchultem

During the course of this lab, I gained a better understanding of what localization in the context of autonomous navigation would look like; from my past experiences, a lot of the autonomous navigation that I had seen focused on more reactive techniques. Knowing that our next lab is focused on path planning, I am curious to see

how these concepts integrate with a more algorithmic process. In terms of the communicative aspect of this lab, with the presentation, I feel that I am definitely learning more about making engaging and clear presentations and am working toward improving the communication of our ideas through the slides. As far as data collection, I did notice that a lot of the questions we receive are about the data collection, so I think that I should help my group more with this in the upcoming lab as it would be beneficial for us as a group and as presenters.