Lab 5: Robot Localization in the Stata Basement using the Monte Carlo Localization Algorithm

Joseph Camacho, Christina Chen, Timothy Kostolansky, Nicole Shigiltchoff, Jessica Wu

I. Introduction

A. Challenge Definition (Nicole)

In previous labs, our racecar used only exteroceptive information collected from its LiDAR and stereo camera sensors to make decisions. This week, our team was tasked with combining exteroceptive LiDAR data with proprioceptive odometry information to determine the location and orientation, together known as the pose, of the robot in a previously mapped location: the Stata basement.

To accurately localize the robot in the Stata basement, both exteroceptive and proprioceptive information is necessary. LiDAR data is useful for determining the shape and distance of the nearby surroundings of the robot and can be compared to the wall and obstacle outlines on the existing map of the Stata basement to determine possible robot poses. However, multiple locations in the Stata basement, as well as in any typical space, have similar features, and some parts of the basement, such as long, smooth hallways, have no notable features that are recognizable to the LiDAR scanner. Therefore, LiDAR information is frequently unhelpful in providing the exact location of the robot—it frequently provides multiple possible locations. In contrast, odometry information, such as the robot's linear and angular velocities, is collected in a manner that is not biased by external features (or lack thereof). Unfortunately, odometry information is rarely accurate, as it cannot account for any asymmetry in the physical structure of the robot or any other external factors that can cause the robot to move not exactly as expected. Odometric estimation of the robot's location quickly accumulates error and can become incredibly inaccurate.

Together, however, the exteroceptive LiDAR data and proprioceptive odometry information can be combined to produce a much more accurate location estimate using the Monte Carlo Localization (MCL) algorithm, a probabilistic approach that can determine the most likely location of the robot by comparing the robot's actual LiDAR scan with the hypothetical scan of all possible positions of the robot. We used the MCL algorithm to accomplish our goal of localizing the robot in the Stata basement.

B. Challenge Motivation (Nicole)

Motivations for this lab included having each of our group's members understand how the MCL algorithm works, as localization is a common problem in many robotics applications and it would be useful for us to know how to address it later. Additionally, we will be using our MCL algorithm implementation to facilitate path planning in lab 6, and we will likely need to implement a localization algorithm in our final project, the race around Johnson Track, in order for our robot to know where it is on the track.

II. Technical Approach

A. Monte Carlo Localization(Nicole, Jessica, and Tim)

Monte Carlo Localization (MCL) initializes a random distribution of possible robot poses, also called particles. The goal of MCL is to localize the particles to a small region on the map in order to estimate the robot's pose on the map. The robot uses both exteroceptive LiDAR data and interoceptive odometry information to do this task.

As the robot moves, its odometry information is used to update the location of each particle (Fig. 1). Its physical environment is also scanned with LiDAR, determining how far away the closest obstacle is and thereby creating an outline of the environment around the robot. Ray casting is then used on the particles to determine what the LiDAR data would look like for each particle, creating environmental outlines for the robot's possible poses. These particle environmental outlines are then compared with the actual environmental outline produced by the LiDAR scan, and the amount of similarity between a particle's projection and actual environmental outline is used to assign likelihood weights to each particle (Fig. 1). More similar, and therefore more likely, particles get assigned proportionally higher weights. With this information, the robot's position is estimated as the weighted mean of the particles (Fig 1).

This process of the robot moving and the MCL particles' location and weights being updated with odometry and LiDAR data comparison, respectively, is repeated until the particles' entropy is below a threshold. Then, the particles are resampled from the weighted distribution of previous particles, and the whole process is repeated continuously as the robot drives through the Stata basement taking measurements about itself and its environment.

B. Initializing Particles (Joseph)

Finding out where a robot is based only on what it can see—and no other information—is known as the Kidnapped Robot Problem and is a very difficult problem.

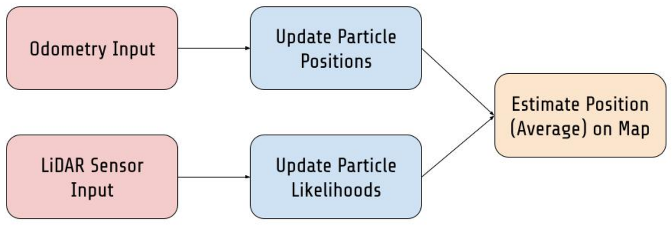

Odometry Input
Update Particle Positions
LiDAR Sensor Input
Update Particle Likelihoods
Estimate Position (Average) on Map

Fig. 1. An illustration of our MCL algorithm. It uses odometry and LiDAR information to update the positions and likelihoods of the particle, and then uses the particles to estimate the average position of the robot on a precomputed map. This estimate increases over time.

Solving this is a very difficult problem, and quite out of scope for this class.

We solved a much easier problem: given a rough initial estimation provided by humans, initialize a set of reasonable potential particles. To sidestep these issues, we provide a rough estimate (within 10 cm) of the robot's initial position and orientation. We can then initialize the particles in a wide circle around this estimate, following a Gaussian distribution.

We chose an initial radial standard deviation of 0.84 m and an initial angular standard deviation of pi / 2 radians, and found in simulation that it could converge to the correct location within 1 second for initial radial errors of up to 1 meter away (Fig. 2). We also tested initial angular errors, but the initial particles' angles were spread out enough that changing the angular error did not measurably affect convergence.

C. Estimating the Next Position of Particles using a Particle Motion Model(Nicole and Jessica

To estimate the position of each particle after the robot moves, a new estimation of particle positions must be calculated using odometry information from the car. Odometry information includes the changes in the car's position and rotation, as well as the duration of time since the last odometry information was logged. Given this information, we determine a transformation matrix to apply to the current particles and calculate new particle positions. The calculation for this particle motion model is as follows:

$$\begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} cos(\theta_{t-1}) & -sin(\theta_{t-1}) & 0 \\ sin(\theta_{t-1}) & cos(\theta_{t-1}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$$

x_{t-1} and y_{t-1} are defined as the previous time step's (x, y) coordinates of a particle, and θ_{t-1} is defined as the previous time step's particle orientation. Δx and Δy are defined as the odometry data from the robot that describe the robot's change in position between the current and previous timestep, and $\Delta\theta$ is defined as the change in orientation. x_t and y_t are defined as the current time step's (x, y) coordinates of the particle, and θ_{t-1} is defined as the current time step's particle orientation.

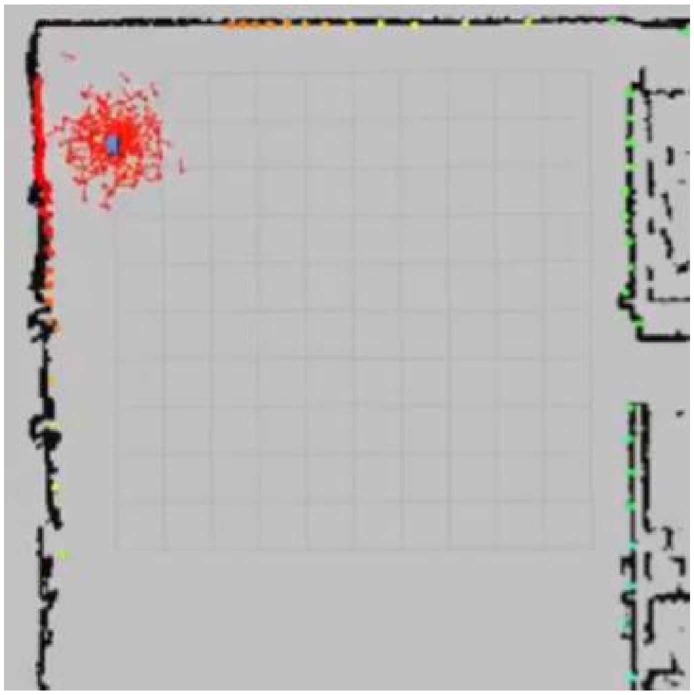


Fig. 2. An example of the particles right after initialization in simulation. The particles, represented by red arrows, are clustered around the car, represented by a blue rectangle. LiDAR data points, represented by points ranging from red to blue depending on their measured distance from the car, will be used to decrease the spread of the particles as the robot moves. The particles will converge to one location, the estimated location of the robot, within 1 second.

D. Computing Particle Likelihoods with the Sensor Model(Jessica and Joseph)

We use a Hokuyo LiDAR scanner to estimate the particle likelihoods. The weight, or likelihood, of a particle in MCL is determined by how likely it is to record LiDAR scan data z_k from a possible position x_k on a previously defined, static map while knowing that the LiDAR scan data at x_k should be d (d is determined through ray casting on the map.)

This likelihood is modeled by four possible cases: the hit case, the short case, the max case, and the random case.

1) Hit case

$$p_{hit} = \begin{cases} \exp(-\frac{(z_k-d)^2}{2\sigma^2} & \text{if } 0 \leq z_k \leq 200 \\ 0 & \text{otherwise} \end{cases}$$

2) Short case

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

3) Max case

$$p_{max} = \begin{cases} 1 & \text{if } z_k = 200 \\ 0 & \text{otherwise} \end{cases}$$

4) Random case

$$p_{rand} = \begin{cases} \frac{1}{200} & \text{if if } 0 \leq z_k \leq 200 \\ 0 & \text{otherwise} \end{cases}$$

All of these cases are then summed into one distribution that accounts for the different likelihoods of each case, weighting each probability case by a set of experimentally determined α values. The final distribution of this likelihood can be calculated as

$$p(z_k|x_k, m) =$$

$$\alpha_{hit} \cdot p_{hit} + \alpha_{short} \cdot p_{short} + \alpha_{max} \cdot p_{max} + \alpha_{rand} \cdot p_{rand}$$

E. Using a Particle Filter to Improve Particle Accuracy

Using this equation, we created a table to compute the likelihoods of all possible particle likelihoods for discretized, integer values of z_k and d between 0 and 200 meters (the possible range of distances produced by the Hokuyo LiDAR scanner on the robot) (Fig. 3). Precomputing and referencing a lookup table of all possible likelihood values with all the possible z_k and d combinations greatly increase the speed of our MCL implementation, as computing the likelihoods has a relatively high time cost and our algorithm is doing this operation at a frequency of 20 Hz.

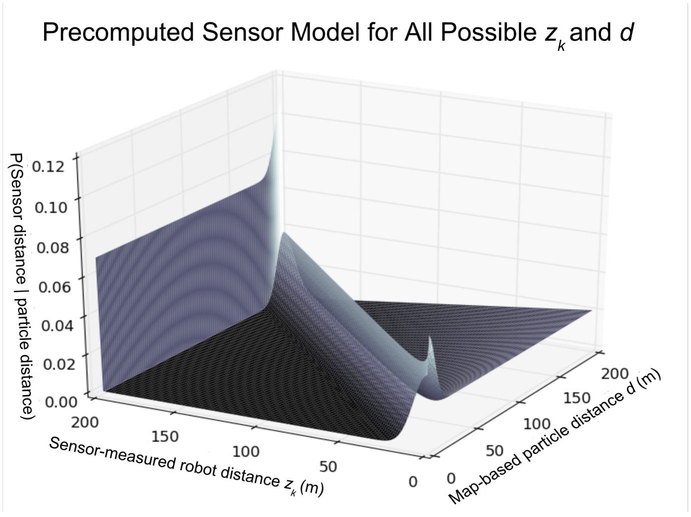


Fig. 3. A plot of the precomputed $p(z_k|x_k, m)$ values vs the sensor-measured robot distance z_k and the map-based particle distance obtained by ray casting d. As expected, the probabilities of the z_k and d being equal are the highest, and the probabilities of the two values being very different are very low, except for $z_k = 200$, the case in which the LiDAR scanner is attempting to measure a distance greater than 200 m, the maximum of its range of measurement.

This conditional probability represents new evidence that we should use to update our particle likelihoods using Bayes rule:

$$P(robotspose = pose_i|lidarscan) =$$

$$P(lidarscan|pose_i)P(robotspose = pose_i),$$

giving the updated rule

$$likelihood_i \leftarrow P(lidarscan|pose_i) \cdot likelihood_i.$$

Unfortunately, this naive update rule fails to take into account that nearby LiDAR scans provide highly correlated information. To correct for this, we can "squash" the conditional probability using the time elapsed since the last scan, making it so that the expected change in log likelihoods does not depend on the number of LiDAR scans taken in a given time:

$$likelihood_i \leftarrow P(lidarscan|pose_i)^{\Delta t} \cdot likelihood_i.$$

Finally, we rescale all the particles' likelihoods so they sum to 1:

$$likelihood_i \leftarrow likelihood_i / \sum_i likelihood_i.$$

F. Determining the Pose of the Robot (Nicole and Joseph

The odometry and LiDAR data update 200 particles at every measurement instance. To localize the car to a location on the map, an estimate of the car's pose must be taken from these particles. To do this, we take an average of the particles.

To determine an estimate of the robot's pose, we separately calculated its location and orientation. The location was determined by taking the weighted mean of the particles' x coordinates for the x position and the weighted mean of the particles' y coordinates for the y position (Fig. 4). These values were considered to be the (x, y) coordinates of the robot. We determined the orientation by taking the circular mean of the particles' relative rotations by projecting their angles onto the unit circle and then taking the mean of the points intersecting with the edge of the circle (Fig. 4).

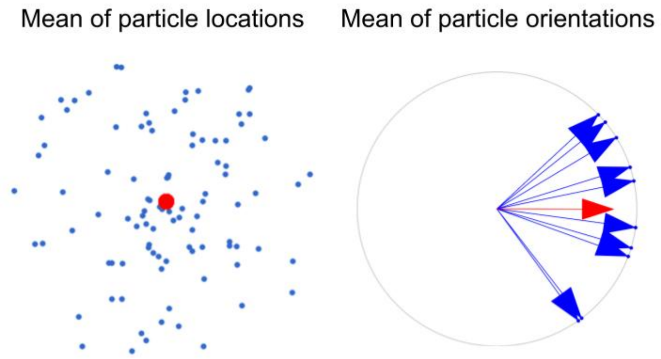


Fig. 4. A representation of our calculations of the location and orientation of the robot. The robot's position, represented by the red point, is determined by taking the mean of the x and y coordinates of every particle, represented in blue. The orientation of the robot, represented by a red arrow, is determined by taking the circular mean of the orientations of every particle, represented in blue.

G. Resampling Particles (Joseph

When determining and updating beliefs about particle positions, particles with higher likelihoods are preferred over those with lower likelihoods. We do this in a step called resampling. During this step, we resample particles using the likelihoods that are calculated in the previous distance sensor (LiDAR) step. Resampling is not done with every likelihood update (i.e., after every LiDAR scan), as is described below.

One issue we faced with our particle filter was a loss of variety in our particles after resampling. Even if the particles are all equally likely, a random resampling of the particles will reduce the number of distinct particles by about 37 percent. (The expected number of particles lost after resampling with equal weights is $n(\frac{n-1}{n})^n \approx \frac{n}{e} \approx 0.3679n$.) This means that if we resample after every LiDAR scan, our particles will collapse exponentially quickly onto a single point. To circumvent this problem, we (1) added Gaussian noise after every resampling and (2) resampled less often by not resampling unless the entropy of the particles (i.e. $-\sum_i likelihood_i \ln(likelihood_i)$) became too small. Our specific choice of threshold was $n \log n/2$, where n was the number of particles in our filter—half the entropy of a uniform distribution of particles. This threshold allowed us to resample when our particles' weights became too uneven (and thus didn't represent the actual distribution very optimally) while avoiding the exponential collapse mentioned above.

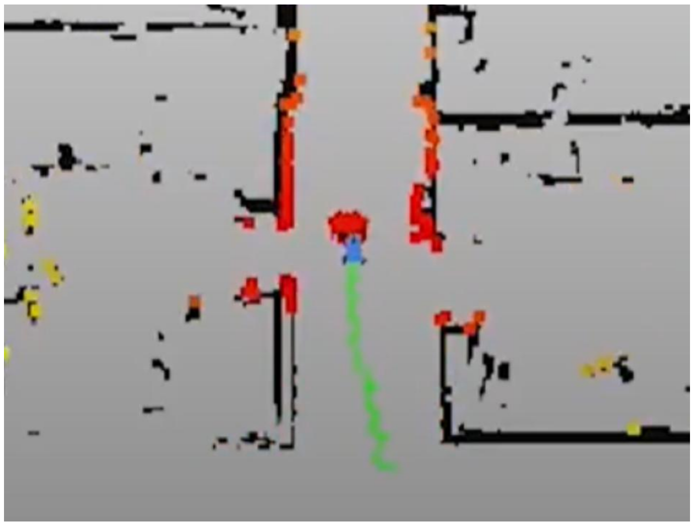


Fig. 5. A visualization of robot localization in simulation. The "slime trail", represented as a green line, shows all of the previous positions of the robot as calculated by our MCL algorithm. Our robot is represented in its position as a blue car with black wheels, and the particles used to estimate the robot's pose are represented as red arrows. LiDAR data points used to update the particles are represented by points ranging from red to blue, depending on their measured distance from the car.

III. Experimental Evaluation

A. MCL Localization in Simulation (Christina)

We first verified our algorithm's functionalities in simulation. We used odometry and LiDAR sensor input to update the participle positions and orientations. That data computes an estimated average position on the map. To visualize this, we created a marker array that identifies all of the robot's historical position data that can leave a "slime trail" of points our robot has traversed according to our MCL algorithm. Later, we upgraded that visualization to be the optimized Path visualization native to ROS and RViz which more smoothly displays where the robot car has been over time (Fig. 5). We were able to visually verify that the position determination is accurate based on how similar the "slime trails" were to the actual path of the car. To run our localization algorithm in simulation, we utilized Joseph's wall follower simulation to move the car in simulation and qualitatively determine if the robot understands its position and orientation.

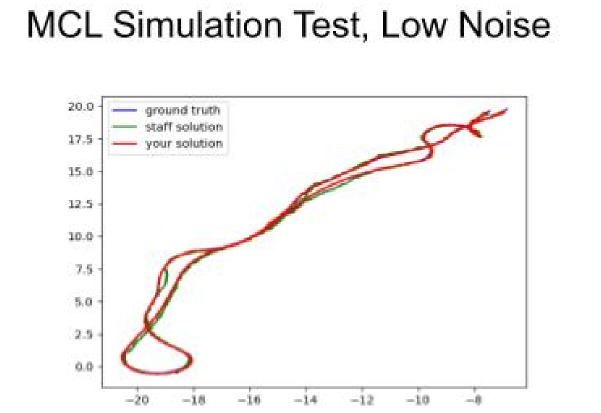
We determined the accuracy of our model by running given givens that compare our results with the staff answers, and our results gained higher accuracy than the staff solution. Fig. 6 illustrates how our model robot's trajectory in simulation with given odometry information compares with the ground truth and the staff implementation. Each graph varies in the amount of gaussian noise in the odometry, and even with varying levels noise, our implementation captures the right trajectory projection.

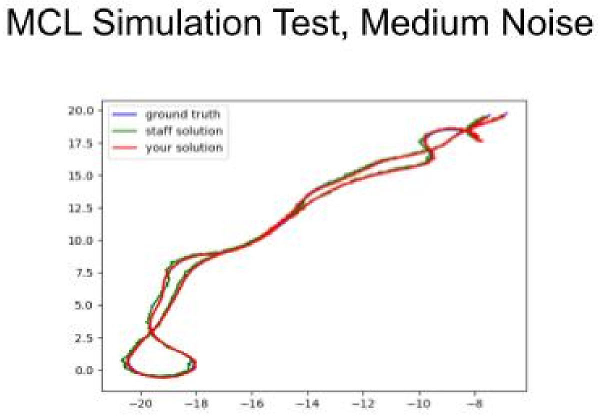
B. MCL Localization in the Physical World (Christina)

Simulation mimics perfect environments, the physical world has randomness and more noise that it needs to account for. To combat this concern and verify our simulation results, we set up experiments in real life, collected data, and analyzed it. Firstly, moving the algorithm from the simulation into the real world required troubleshooting and adjusting some parameters. For example, setting up the localization package required initializing poses manually. The racecar computer does not allow publishing of the initial poses to happen, so we need to collect that data on our local computer and feed it into the racecar computer, so our initial pose is accurate. We also needed to add a lock on our code, so that parameters are not initialized incorrectly due to race conditions. There was significant lag and threading problems in our odometry and lidar data collection. The odometry data collection process was more delayed in receiving and populating data, so the odometry data collection and publication would populate our algorithm with outdated information, and the algorithm would not be able to update since the data is lost. We implemented a lock on the data population, and that elevated the problems.

After troubleshooting, we were able to control the racecar and see it locating its position and orientation live. We can visibly see that the robot was able to correct its orientation on the Rviz as we modified the angle that it was facing. We can quantitatively see our algorithm with its particles at work via the following graphs. Fig. 7 depicts how the particles had a high spread initially, but quickly converges to a smaller standard deviation value.

When our particles eventually converged, we initially encountered cases when our particles converged to an incorrect location, and the algorithm was not able to allow the particles to converge into the correct location.





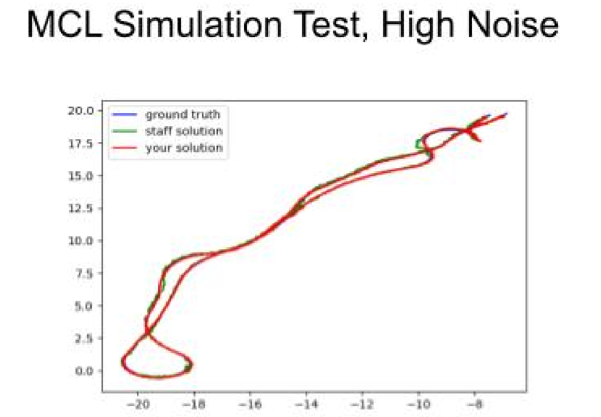


Fig. 6. Localization results from simulation tests of our MCL algorithm, with low, medium, and high gaussian noise. These tests were generously provided by the 6.4200 teaching staff. For each level of noise, our simulation performed better than the staff solution by following the ground truth path more closely and more smoothly.

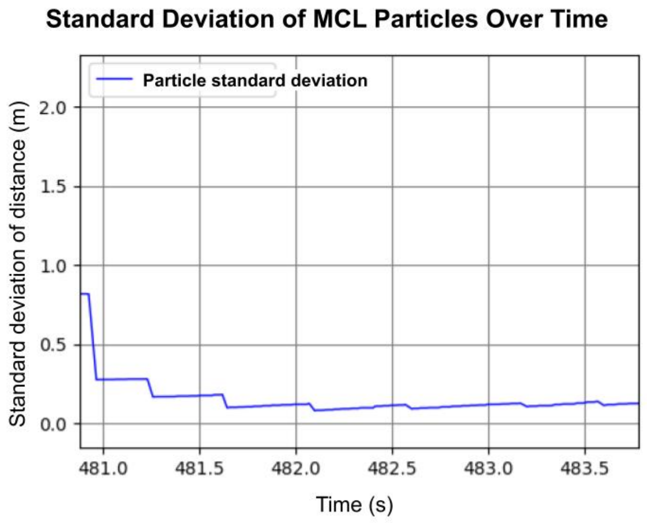


Fig. 7. A plot of the convergence of the MCL's particle spread, showing the standard deviation of the particle cluster over time. Initially, the standard deviation is 0.84 m, but decreases below 0.15 m within a second and stays low

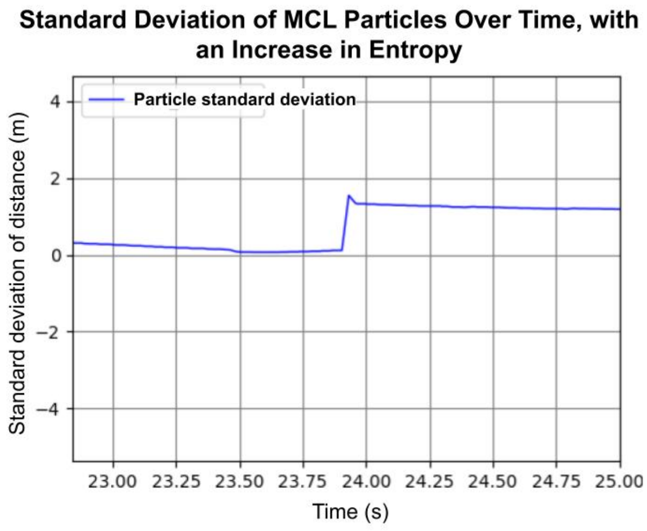


Fig. 8. A plot of the convergence of the MCL's particle spread, showing the standard deviation of the particle cluster over time. Between 23.75 and 24.00 s, the entropy of the particles become lower than $nlog(n)$ (with n representing the number of particles), so our data is resampled to have the initial sampling standard deviation (in this case, the initial standard deviation was set to 1.75 m).

To address this, we decided to resample our data at a threshold when entropy is low (lower than $nlog(n)$, with n representing the number of particles), so our new particles can have a larger spread and converge again to a different location (Fig. 8).

IV. Conclusion (Nicole and Christina

In lab 5, we successfully implemented the Monte Carlo Localization (MCL) algorithm on our robot in multiple simulated environments, as well as in the physical world. Using LiDAR and odometry data, our algorithm was able to narrow down our spread of possible particle locations and orientations to a pose to represent that of the robot. Our results were accurate, as evident by a visual comparison of the robot's pose in simulation

or the physical world and the particles' coordinates. In situations when the initial convergence of the particles was not consistent with the robot's location, this was able to be resolved with the help of resampling any time the particles' entropy got too low. Our implementation allows for our robot to traverse the Stata basement while using LiDAR and odometry data to determine where it is.
Though we accomplished our goal of localizing our robot in the Stata basement with the help of MCL, there are still aspects of our implementation that we can improve. Our current method for determining the actual pose of the robot with information about our particle cases works well for a unimodal distribution of particles. However, this method requires taking the mean of particles' positions and orientations, which would fail in the case of a multimodal distribution of particles. If our MCL narrowed down the possible pose of the robot to two or more possible places on a map, our algorithm would place the robot in the middle of the two-particle clusters representing the possible locations. This estimate is incorrect, as our robot pose estimate only determines a single average position over all the particles. We did not run into this issue while testing in any simulated map or the Stata basement, but this is likely because the locations where we were able to test have enough features to look distinct from every other location on the map. Despite this, it would be useful to improve our handling of multimodal particle clustering, as we will be using our localization algorithm in lab 6.
Additionally, though our robot was capable of correctly implementing the MCL algorithm in simulation and the physical world, the accuracy in the simulation was higher than in robot tests in the physical Stata basement. Our MCL algorithm would sometimes acquire so much error that the localization was incorrect—our robot was nowhere near the position calculated on the map. This is likely due to a lack of robustness in our algorithm and the fact that the physical environment of the Stata basement is not identical to the predetermined map the robot is using for localization—there were people and furniture in the basement that were not on the map while we ran our physical tests, and all the doors leading out of the main hallway of the basement were closed during our physical tests while they were open in the predetermined map. Due to time constraints during this lab, we were not able to fully explore solutions to increase robustness to different types of naturalistic noise. While we did include Gaussian noise in our MCL simulations to which our algorithm was robust, it would be beneficial to add different kinds of noise and improve our algorithm's robustness to those in the future.

V. Lessons Learned

1) Joseph: I learned how to downsample data using cubic splines. I also learned how to apply Bayesian reasoning in engineering! Finally, between my four final projects and two midterms, I learned how to manage my time effectively and quickly switch between different tasks.

2) Christina: Technically, I understood how to implement a localization algorithm like MCL to utilize statistics and sampling to determine a robot's location and orientation. I learned effective bug isolating skills and asked for help to pinpoint problems like the incapability of initializing particles on the racecar's computer faster. I'm still learning how to work effectively in a team with varying skills and interests, but we have the groundwork of trust and care to continue improving our teamwork.

3) Timothy: I learned how to localize a robot's position on a predetermined map using the MCL algorithm. I also learned how to deal with issues of filter robustness, as original implementations of our particle filter converged quickly and often to erroneous poses. I also improved my testing and iteration skills, as this is necessary to test small updates to our algorithm.

4) Nicole: I learned about and helped implement the Monte Carlo algorithm on a physical car, which was exciting because localization is a common and useful task that many robots need to accomplish. I also did some thinking independently about how algorithmic localization works in comparison to my own, organic localization, and it was cool to discover some of the similarities and differences between how those two systems can work. Additionally, I gained more experience trusting my teammates to accomplish what was necessary even if I wasn't involved, as this lab had a lot of components that were created in parallel and were then connected later.

5) Jessica: In this lab I learned about real-world applications of the Monte Carlo algorithm, which also means I gained a deeper understanding of the probability and statistical concepts used, such as random sampling and data interpretation. I think my team also learned a few valuable lessons on efficiently splitting up tasks among us and the importance of being understanding and empathetic towards each other.