Lab 5 Report: Localization

Team 16: Sameen Ahmad, Phillip Johnson, Trevor Johst, Maxwell Zetina-Jimenez, Ellen Zhang

1 Introduction

Author: Sameen Ahmad

Localization is a complex challenge in robotics as it often involves uncertainty, unknown environments, and balancing computational resources. Nonetheless, it is a fundamental capability that underlies many aspects of robot functionality, enabling robots to perceive, understand, and interact with their environment effectively.

The goal of this lab was to develop and implement an algorithm that allows our autonomous race car to determine its pose (orientation and position) relative to the environment. Specifically, our focus was integrating the Monte Carlo Localization algorithm on our racecar by leveraging LIDAR and odometry sensor data.

This directly builds upon our previous work of utilizing LIDAR when we integrated our wall follower algorithm. With this implementation, our next steps involve applying information about the robot's location towards path planning and in making decisions about future actions. This lab fits into our broader goal of developing a robust autonomous racecar capable of racing against other robots on the Johnson Track.

The Monte Carlo Localization algorithm uses a collection of 'particles' to estimate the pose of a robot. Particles, which represent a potential state of the robot, are initialized at a location. Through the motion model, the pose of the particles are updated to reflect possible future states. The robot uses LIDAR to receive feedback on the environment. The sensor model determines the likelihood that each particle received the range sensor reading and assigns corresponding weights. Particles are then resampled and eventually converge on the expected pose of the robot.

We began by separately implementing and testing the motion model and sensor model. Then, we integrated the components to create our particle filter. We tested and refined our implementations in the racecar simulation, before adapting our solution for our final integration on the racecar.

Section Editor: Trevor Johst

2 Technical Approach

Author: Sameen Ahmad

Our development of the particle filter involved implementing the motion model and sensor model independently before integrating it virtually and physically on our race car. The motion model relies on using odometry to generate and continuously update potential poses of the robot, while accounting for noise. Whenever LIDAR data is received, the range values are down sampled and discretized to improve computational efficiency. The sensor model then loops through each particle and determines the likelihood that it represents the state of the robot. Based on the particles with the highest weight, the robot is able to update its estimated position. Particles are resampled and redistributed in proportion to their weights, where those with a higher probability appear in greater frequency than others. This processes repeats causing the particles to converge on the robot's true position.

Section Editor: Trevor Johst

2.1 Motion Model

Author: Trevor Johst

Odometry data from the onboard Inertial Measurement Unit (IMU) provides information on the instantaneous change in pose at a rate of 50 Hz. An accumulation of small errors over time will cause this odometry to deviate from reality. By intentionally adding noise to our readings, the motion model can replicate natural uncertainty in movement of the robot. Then, the sensor model compares the pose of each particle with feedback

The effect of this noise can be seen in Fig. 1, where our potential locations for the robot spread out as we go longer without an update. By itself the motion model would not assist in localization, but other sensor data allows this connection to occur. As the motion model causes particles to drift, it encapsulates all potential poses of the robot. A sensor model can then select which poses are most likely.

2.1.1 Theory

Formally, the goal of the motion model is to calculate the robot's pose, x_k, as a function of the previous pose, x_{k-1}, and odometry readings, u_k. This can be written as

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

Since the IMU only provides odometry in terms of instantaneous velocity, it must be integrated before updating position. It should be noted that by integrating a potentially erroneous sensor reading, we are compounding the error with any previously experienced error. The change between readings can be expressed as

$$\mathbf{\Delta x_k} = \mathbf{u_k} \Delta t \tag{2}$$

Additionally, these readings will be in the reference frame of the robot. A rotation matrix must then be used to transform into the world frame. A full update step can be represented as

$$\mathbf{x_k} = \mathbf{x_{k-1}} + \mathbf{R} \mathbf{\Delta x_k} \tag{3}$$

$$\mathbf{x_k} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \cos(\theta_{k-1}) & -\sin(\theta_{k-1}) & 0 \\ \sin(\theta_{k-1}) & \cos(\theta_{k-1}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta y_k \\ \Delta \theta_k \end{bmatrix} \tag{4}$$

If the readings from the odometry were perfect, (4) would be sufficient to localize the robot given an initial pose. However, the aforementioned accumulation of error means this uncertainty must be mathematically represented. We

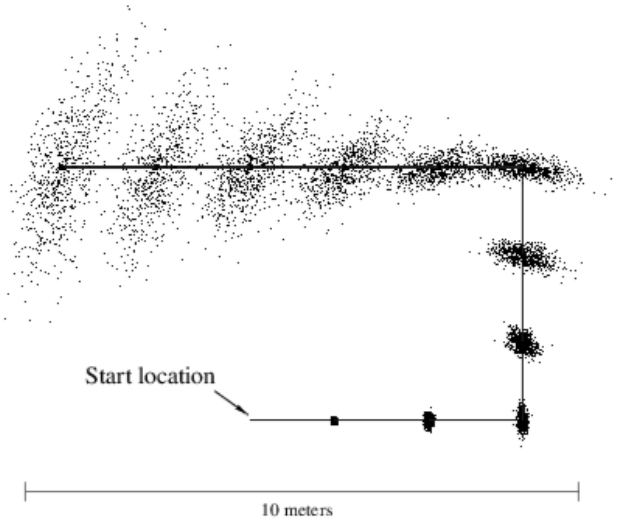

Fig. 1. **Particles spreading out as a robot travels along a set path.** The solid line represents the ground truth trajectory of the robot, and the dots represent potential positions when accounting for noise. The longer the robot goes without receiving sensor data, the less certain we are about its position.
Source: [1]

decided to represent it through the addition of a Gaussian distribution, as this is typically the profile found on IMUs [2]. A Gaussian distribution is expressed as

$$\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \tag{5}$$

where σ is the standard deviation and μ is the mean. Let ϵ represent a specific sample from the distribution. We simply add our noise to the pose, meaning we can generate it with a μ of 0. The values of σ were determined through experimentation to balance a proper spreading of particles and sufficient convergence.

Choosing to apply our noise after the update step effectively introduces some random translation and rotation. Had it been applied directly to the odometry, the rotation matrix would compound the noise on the translations but not on the angle. The noise could additionally be scaled by Δt to ensure longer timesteps account for more possibility of drift.

If we choose to view this noise as an input into our motion model, we can finally represent the full model as

$$\mathbf{x_k} = f(\mathbf{x_{k-1}}, \mathbf{u_k}, \boldsymbol{\epsilon}) \tag{6}$$

$$\mathbf{x_k} = \mathbf{x_{k-1}} + \mathbf{R u_k} \Delta t + \boldsymbol{\epsilon} \tag{7}$$

2.1.2 ROS2 Implementation

As the robot travels, it continuously publishes odometry data from the IMU. A listener on this topic then updates every particle's position and introduces a small amount of noise, as outlined in Algorithm 1.

To complete our full update step with noise, we only need to acquire Δt and $\boldsymbol{\epsilon}$. The ROS2 clock allows us to keep track of the time between loops and determine how much has elapsed. Our noise vector is generated by sampling from our Gaussian distribution using `np.random.normal`.

Every particle must then be looped over and updated independently in a "for" loop. The only constant across all particles will be Δt.

Once all of the particles are updated, the only remaining step is to also update our estimated position. Since the odometry itself represents our best possible guess for the transformation of the robot, this update is executed without noise.

Algorithm 1 Motion Model

$\Delta t = prevTime - curTime$
$\Delta x = u_k \times \Delta t$
$updated = [\,]$
for x_{k-1} **in** particles **do**
  $R = \text{ROTATIONMATRIX}(x_{k-1})$
  $x_k = x_{k-1} + R \times \Delta x$
  $\epsilon = \text{GAUSSIAN}(\sigma)$
  $x_k = x_k + \epsilon$
  $updated \leftarrow x_k$
end for
$prevTime = curTime$
return $updated$

Section Editor: Maxwell Zetina-Jimenez

2.2 Sensor Model

Author: Maxwell Zetina-Jimenez

Sensor data can be incredibly useful to understand the robot's position relative to its environment. The goal of the sensor model is to evaluate how probable a sensor reading z_k from a particular pose x_k is at some discrete timestep k from a given map m. In other words, we hope to find

$$p(z_k|x_k, m) \tag{8}$$

The sensor model thus helps find which poses are most likely to produce a sensor reading like the one that was measured. Probabilities are assigned to each pose with the goal that better pose estimates are more likely to be resampled at the next time step.

2.2.1 Theory

The probability of a sensor reading z_k given a pose x_k at time k is dependent on the type of sensor. In this lab, a LIDAR scanner was used for sensor readings. Thus, the probability of a particular scan z_k is the product of the probabilities of each beam measurement $z_k^{(i)}$ from the n beams in that laser scan:

$$p(z_k|x_k, m) = \prod_{i=1}^{n} p\left(z_k^{(i)}|x_k, m\right) \tag{9}$$

The calculation for a scan thus relies on the calculation for a beam measurement. However, to account for the different possible results a laser beam measurement could have, this is determined from the probabilities of the following cases:

1) $p_{hit}\left(z_k^{(i)}|x_k, m\right)$: Detecting a known map landmark
2) $p_{short}\left(z_k^{(i)}|x_k, m\right)$: Performing a short measurement due to a close obstruction

3) $p_{max}\left(z_k^{(i)}|x_k,m\right)$: Performing a large measurement due to the laser not being returned
4) $p_{rand}\left(z_k^{(i)}|x_k,m\right)$: Performing a random measurement due to an unexpected event

Since p_{hit} indicates a match with a known landmark, we would like this probability to be the highest of the four. For this reason, p_{hit} is calculated with a Gaussian distribution that is centered around the true distance d from the pose to the landmark – if $z_k^{(i)}$ matches d, then this will be maximum:

$$p_{hit}\left(z_k^{(i)}|x_k,m\right) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{(z_k^{(i)}-d)^2}{2\sigma^2}) & 0 \le z_k^{(i)} \le z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Close obstructions in the second case are more likely to be picked up by LIDAR when they are closer to the robot (assuming they are uniformly distributed in the environment). Thus, we represent this case as a decreasing function:

$$p_{short}\left(z_k^{(i)}|x_k,m\right) = \begin{cases} \frac{2}{d}(1-\frac{z_k^{(i)}}{d}) & 0 \le z_k^{(i)} \le d \ne 0 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

Moreover, the third case represents a large measurement near the maximum sensor reading. We would like to approximate the difference between the high measured reading $z_k^{(i)}$ and the maximum reading z_{max} with the following model, using a small factor ϵ:

$$p_{max}\left(z_k^{(i)}|x_k,m\right) = \begin{cases} \frac{1}{\epsilon} & z_{max}-\epsilon \le z_k^{(i)} \le z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Finally, a random measurement (fourth case) should have a small impact on the outcome due to its unlikelihood:

$$p_{rand}\left(z_k^{(i)}|x_k,m\right) = \begin{cases} \frac{1}{z_{max}} & 0 \le z_k^{(i)} \le z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

To produce the final probability for a particular beam measurement, these four probabilities are then joined in a weighted average:

$$\begin{aligned} p\left(z_k^{(i)}|x_k,m\right) = \alpha_{hit}\cdot p_{hit}\left(z_k^{(i)}|x_k,m\right) \\ +\alpha_{short}\cdot p_{short}\left(z_k^{(i)}|x_k,m\right) \\ +\alpha_{max}\cdot p_{max}\left(z_k^{(i)}|x_k,m\right) \\ +\alpha_{rand}\cdot p_{rand}\left(z_k^{(i)}|x_k,m\right) \end{aligned} \tag{14}$$

where α_{hit}, α_{short}, α_{max}, and α_{rand} represent the respective weights of each case (and sum to 1 so that $p(z_k^{(i)}|x_k,m)$ forms a valid probability distribution).

With this calculated probability for a particular beam measurement in a scan, it is possible to then compute $p(z_k|x_k,m)$, the probability for the whole laser scan data for a particular pose x_k. This is done for all particles – potential poses – to update the probabilities for each, which is used to filter poor pose estimates and favor probable ones.

2.2.2 ROS2 Implementation

Following the theory behind the sensor model, the implementation was similar. A key difference was that calculating all of the probabilities for a whole scan for each particle pose is computationally expensive and time-consuming.

Rather than doing these operations with every exact measurement $z_k^{(i)}$, a discretized grid of possible combinations of measured distance and actual distance (from pose to object) was used to store precomputed probabilities for a respective $(z_k^{(i)}, d)$ pair (that was discretized to be able to be retrieved from the precomputed table). This allowed for values of p_{hit}, p_{short}, p_{max}, and p_{rand} to be quickly computed (following their respective equations) from a lookup table that represented the sensor model and thus more efficiently assigns probability weights to poses.

Because of this discretization, however, ϵ was set to 1 in (12). Finally, because each column of the table corresponded to a possible true distance d, each column was normalized so that the probabilities along each column summed to 1, thus forming a valid probability distribution.

With this table, the implementation followed the theory to compute probabilities for potential poses with Algorithm 2, which grabs the probability for each (beam, true-distance) pair from the lookup table and computes the product of all pairs as in (8) to determine the probability for that particle pose. The true distance comes from the simulated scan for each particle (i.e. the simulated laser scan if the robot were at that particle pose). It does this for every particle to get all of the updated particle probabilities.

Finally, it normalizes the probabilities across all particles so that they sum to 1 and the probabilities of the particle poses form a valid probability distribution. With a probability assigned to each particle, the sensor model and motion model can now be integrated together to produce a hypothesis pose for the robot.

Algorithm 2 Sensor Model

```
probabilities = [ ]
for sim scan in particle scans do
    weight = 1
    for (z_k^(i), d) in (measured scan, sim scan) do
        weight = weight * LOOKUPTABLE(z_k^(i), d)
    end for
    probabilities ← weight
end for
return probabilities / SUM(probabilities)
```

Section Editor: Phillip Johnson

2.3 Integration

Author: Ellen Zhang

The motion model and sensor model play key roles in the Monte Carlo Localization algorithm, known as particle filtering. In each iteration of the algorithm, the estimated pose and orientation of the robot are updated as it moves and senses its environment, the Stata basement.

2.3.1 Theory

To start with, several particles are initialized randomly around the robot's starting pose. Each of these particles

represents a guess as to where the robot is at that particular time. When the robot moves around in real life, the program receives real-time odometry and LIDAR information. The particle filtering algorithm goes through three steps:

1) **Prediction** Starting from a set of particles at time $k-1$, pass in the odometry information to the motion model to obtain a new set of particles at time k.
2) **Update** Reassign likelihoods to each new particle using the sensor model.
3) **Resampling** Resample the particles with probability proportional to their likelihood, which helps converge the particles towards the robot's true position.

These three steps are repeated as the robot moves and receives new data, thus the particles are continuously updated to track the movement; Fig. 2 shows the overall structure of the algorithm.

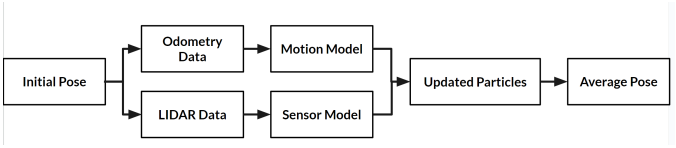

Fig. 2. **Diagram of the flow of Monte Carlo Localization** Starting with the initial pose, the program receives odometry and LIDAR data, which it passes into the motion and sensor model respectively. The particles are updated and used to calculate the weighted average, which is published at a rate of 20 Hz.

2.3.2 ROS2 Implementation

In ROS2, the particle filter works through a system of subscribers and publishers and depends on chosen parameters. Upon program initialization, around 200 particles are initialized around the robot's starting pose, which is manually set using pose estimate in RVIZ. We chose 200 particles because this number achieves a good balance between efficient runtime and having enough particles to work with.

The program subscribes to odometry and LIDAR messages on the car. Every time an odometry message is received, the motion model updates the particle movements. Every time a LIDAR message is received, the LIDAR data is first downsampled from 1081 to 99 rays to avoid redundancy and improve runtime. The downsampled observation is passed into the sensor model to resample the particles according to weights.

The weighted average pose is continuously updated as well. The program publishes the weighted average pose at a frequency of 20 Hz, to ensure real-time performance. Given the array of particles, the average x and y values are calculated as

$$(\bar{x}, \bar{y}) = \sum_{i=1}^{n} p_i \cdot (x_i, y_i). \tag{15}$$

The average θ must be calculated using circular mean,

$$\bar{\theta} = \tan^{-1}\left(\frac{\sum_{i=1}^{n} p_i \cdot \sin\theta_i}{\sum_{i=1}^{n} p_i \cdot \cos\theta_i}\right). \tag{16}$$

Over time, we have seen that the weighted average pose consistently tracks the true location of the car.

Section Editor: Trevor Johst

3 Experimental Evaluation

Author: Phillip Johnson

The integration of the motion and sensor models has generated an efficient and robust algorithm for global localization of the robot. In this section, the integration of the models will be qualitatively and quantitatively analyzed to discuss the performance and potential improvements of the integrated model.

To evaluate the effectiveness and robustness of the model, tests were conducted on accuracy and robustness. For the purpose of this lab, accuracy will be defined on the distance between the estimated position and actual position where position is measured relative to the map origin. Robustness is defined as the ability for a high accuracy to be reached quickly. Convergence and runtime will be evaluated to ensure that the model reaches high accuracy with an ability to update its position estimate at speeds of at least 20 Hz.

Section Editor: Ellen Zhang

3.1 Simulation

Author: Phillip Johnson

Although real-world implementation often requires different parameters than the simulation environment, testing in the simulation will ensure that our algorithms work under best-case conditions while ensuring the safety of the racecar.

3.1.1 Accuracy Evaluation

One luxury in simulation, which we do not have in the real world, is the ability to access the robot's position at all times using a transform lookup between the `/map` and `/base_link` frames. Leveraging this, an error plot (shown in Fig. 3) can be created between the estimated pose and the actual pose relative to the map origin. To calculate the error we used

$$Error\% = 100 * \frac{Estimated - Actual}{Actual} \tag{17}$$

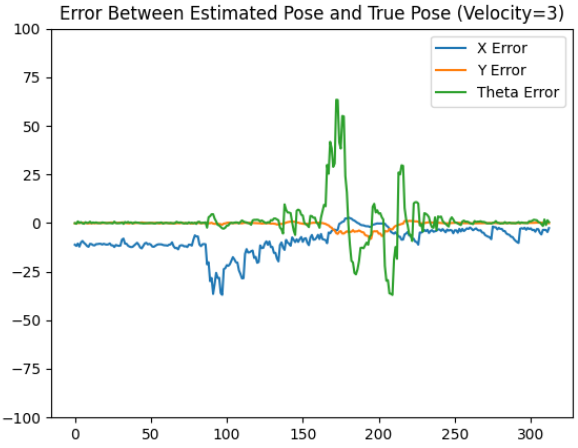

Fig. 3. **Error percentage between actual pose and estimated pose.** Over time, the X and Y errors remain relatively consistent at around 0. For θ, the peak occurs when the car turns a corner, but it quickly converges back to the true θ.

As seen in Fig. 3, the X (vertical) and Y (horizontal) errors remain extremely low even at a velocity of 3 m/s. The θ (orientation) error peaks quite high when the car is turning around a corner, but it eventually converges back toward zero.

Because the error is extremely low in straight hallways and converges back toward zero after spiking, it can be concluded that the localization is robust enough to use in the real-world.

3.1.2 *Runtime Evaluation*

The generated laser scan data from the simulation is roughly 10% the size of the LIDAR laser scan the car would measure in real life. Because of this, it is imperative that the odometry and laser scan callbacks can perform computations at speeds well above 20 Hz. To test this, a timer was set at the beginning of the callback function and ended once the callback was finished. The results were plotted as can be seen below.

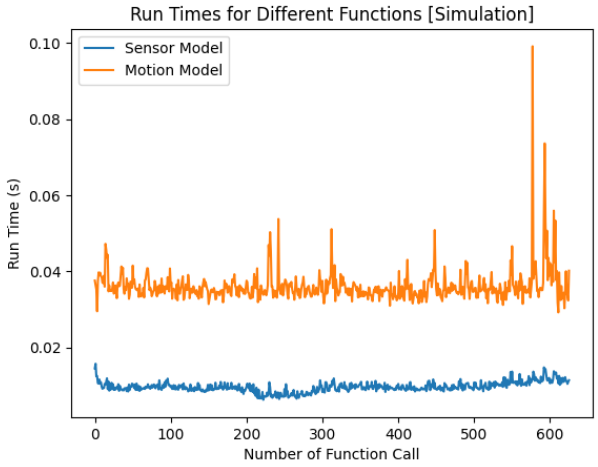

Fig. 4. **Runtime of callback functions in simulation.** The x-axis is the number of function calls and is plotted against the runtime of the function in the y-axis. Note that both the sensor and motion model lie below 0.05 seconds, meaning the program runs efficiently.

To reach the 20 Hz threshold, both models should run in less than 0.05 seconds. As shown Fig. 4, this was achieved in simulation. It is worth noting, however, that computations are largely computer-based and may not represent performance on the robot. The analysis is useful, however, to understand which parts of the code are bottlenecks in computation.

Section Editor: Ellen Zhang

3.2 Physical Implementation

Author: Phillip Johnson

When testing the car in real life, we perform an accuracy evaluation of the algorithm's correctness. Then, we evaluate the efficiency of the algorithm when performing in real time.

3.2.1 *Accuracy Evaluation*

Without the luxury of knowing the robot's position at all times, some creative work must be done to ensure model accuracy. To test the accuracy, three tests were performed:

1) Accuracy around the initial position: since the initial position is known, the estimated position can be easily tested against the initial position.
2) Accuracy along a known distance: knowing the distance and initial position should ensure that the bot ends up at a known location which the estimation can be tested against.
3) Qualitative testing: Viewing the robot's position in simulation and comparing it to the real-world position qualitatively can ensure that no major estimation errors are committed.

In Fig. 5, the results of the known initial test can be seen.

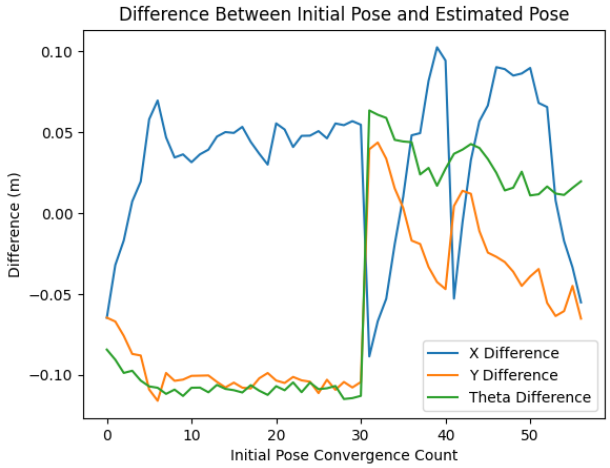

Fig. 5. **Difference between initial and estimated pose.** When the car is initialized, the algorithm quickly locates its estimated position in the map and converges accurately.

From this test, it is clear that the bot can confidently converge to the initial pose with accuracy of around 0.01 m.

For the known distance test, the robot was set up in a hallway that runs along the -x direction and was moved approximately 3.11 meters in the -x direction.

The initial position was X = -8.27 m, Y = 26.41 m, and θ = 3.097 rad, and the final position was X = -11.65 m, Y = 26.53 m, and θ = -3.13 rad. This test showed that the robot was able to track itself confidently along a straight line. Future testing will involve tracking around corners and non-linear motion.

Finally, conducting qualitative analysis showed that the robot's movement in real life was consistent with the estimated position being published in RViZ. Through these three tests, there is strong confidence in the bot's ability to localize with strong accuracy.

3.2.2 *Runtime Evaluation*

Without quick localization, the bot will be unable to run at effective racing speeds. Thus, the goal was for the bot to be able to converge on a point rapidly and publish the odometry vector at a minimum speed of 20 Hz when run in real life.

In Fig. 6, the runtime of both of the callback functions can be observed as the robot drove around.

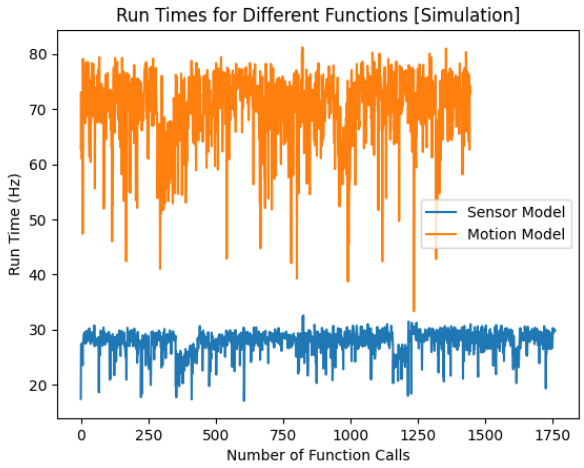

Fig. 6. **Runtime of odometry and laser scan callbacks.** The x-axis is the number of function calls and is plotted against the runtime of the function in the y-axis in hertz. Note that both the sensor and motion model lie above 20 Hz, meaning the program runs efficiently.

As shown in Fig. 6, the sensor model and motion model are both able to consistently update odometry data at speeds above 20 Hz. Additionally, Fig. 7 shows the convergence rate on the initial position.

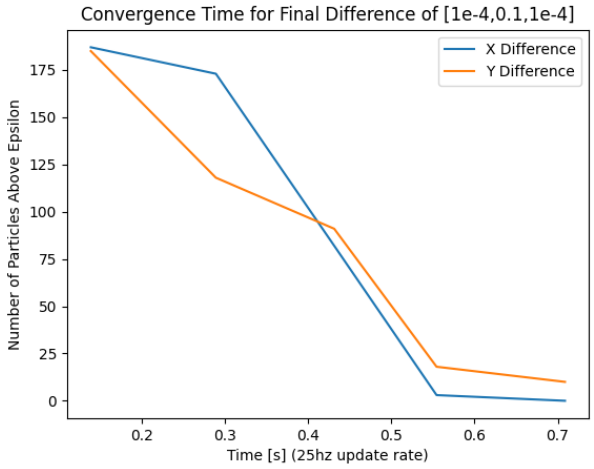

Fig. 7. **Convergence on the initial pose.** At a 25 Hz update rate, the number of particles that are far away from the robot's true position rapidly falls to nearly 0, indicating a robust convergence of the particle filter.

In Fig. 7, it can be observed that the number of particles outside of the epsilon value (0.1 m in this case) decreases rapidly through each function call. An extremely accurate odometry vector can be found in less than a second, but a sufficiently accurate vector can be found in under 0.5 seconds. It is worth noting that this convergence rate is significantly faster once the robot has initialized as less sampling will be required.

Section Editor: Ellen Zhang

4 Conclusion

Author: Ellen Zhang

In this lab, we implemented the Monte Carlo Localization algorithm on our autonomous racecar, utilizing sensor data from LIDAR and odometry to estimate the car's pose relative to its environment. Through the design and optimization of the motion model and sensor model, we successfully created a robust algorithm capable of accurate localization.

Our experimental evaluation in both simulation and physical implementation showcases the effectiveness and efficiency of our approach. In simulation, we achieved low error rates even at high velocities, demonstrating the robustness of our localization system. Additionally, our algorithm meets real-time requirements, crucial for practical deployment on the racecar, especially in the next lab where we will work on path planning.

Looking forward, the successful integration of the Monte Carlo Localization algorithm lays the foundation for further development of our autonomous racecar, paving the way for enhancing our robot's capabilities in navigation and decision-making on the Johnson Track.

Section Editor: Sameen Ahmad

5 Lessons Learned

5.1 Sameen Ahmad

This lab allowed me to familiarize myself with localization algorithms. I was also exposed to advanced programming techniques including multi-threading and profiling during its integration. This lab, however, was the most challenging for me in understanding the Monte Carlo localization algorithm and its implementation as a whole. Although, through debugging I was able to get a better grasp of the material. I also recognized the importance of communication in maintaining the team's momentum. I learned that I shouldn't hesitate to reach out to my team members about my confusions early on, instead of struggling to understand it on my own.

5.2 Phillip Johnson

This lab introduced me to the difficulty of localization and how it can be done effectively. The first lesson that I learned is that noise is tough to deal with. Without a known initial position, localization is near-impossible. Secondly, I learned that collaboration is essential. Everyone on this team contributed to clearing up muddy topics, and the collaboration that we had helped me both finish my tasks and understand what I was doing.

5.3 Trevor Johst

This lab taught me the concept of localization. Additionally, it taught me that random sampling and Monte Carlo approaches can be extremely powerful for solving a complicated issue in a short amount of time. Collaboration is extremely important when working under tight time constraints, and ensuring efficient use of time is often the biggest challenge. Everyone communicating about what issues they are running into, and getting help when needed only makes everything work more effectively.

5.4 Maxwell Zetina-Jimenez

This lab was great to learn more about localization in robotics. A big takeaway for me was the benefit that noise

can have on calculating predictions. I also enjoyed implementing and seeing how a probabilistic algorithm proved to be effective. Going from theory to implementation was a big help in understand the probabilistic models, and I think this was a good technical skill I learned. I also learned that communication plays a big role in collaborative tasks. Being responsive and transparent with roadblocks, needing help, and progress allows for cohesiveness in the team.

5.5 Ellen Zhang

This lab provided me with valuable insights about Monte Carlo Localization, especially in practice when implementing the algorithm on the racecar in real life. Collaboration within the team was essential for understanding and overcoming challenges encountered during the implementation process. This experience helped me realize the importance of communicating tasks ahead of time, and working together to solve problems.

REFERENCES

[1] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," in *AAAI/IAAI*, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:2272361

[2] N. K., S. A. G., J. Mathew, M. Sarpotdar, A. Suresh, A. Prakash, M. Safonova, and J. Murthy, "Noise modeling and analysis of an imu-based attitude sensor: Improvement of performance by filtering and sensor fusion," *SPIE Proceedings*, Jul 2016.