# Lab 5 Report: Localization

Team #10

Diego Delarue
Jack Lewis
Jason Lee
Vlada Petrusenko

RSS (6.4200)

April 15, 2023

## 1 Introduction (author: Diego Delarue)

After implementing the parking controller, the next big challenge in order to reach our goal of path planning is to figure out the car's location and orientation. In order to do this, we are going to ahve to use localization techniques.

This is the information we have at our disposal:

1. LiDAR scans
2. Ground truth map
3. Odometry data from IMU

Using this information, we can locate where we are in the map.
To obtain the car's location and orientation in the map we will implement Monte Carlo Localization (MCL or particle filter).

This implementation gets broken down into 3 steps:

1. Motion Model
2. Sensor Model
3. Particle Filter

The motion model consists of obtaining the odometry of the next location given he current $(x, y)$ location as well as orientation $\theta$. Using the motion changes we know we are sending to the car, we can predict the next location at time $k$ by

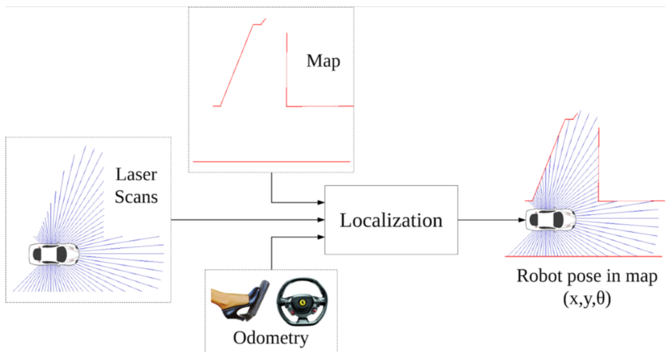


Figure 1: Visual Representation of inputs to Localization.

using the motion model $f(x_{k-1}, \Delta x) = x_k$.

The sensor model consists of a function that takes in generated particles and the LiDAR scan data. The sensor model is in charge of computing at time $k$ the probability of getting a certain reading $z_k$ assuming that we are at location $x_k$ at map $m$, $p(z_k^{(i)}|x_k, m)$.

Finally, the particle filter integrates the two previous functions by simulating particles that assume positions $z_k$ in order to be able to decide which scans from the car match the closest scans from the particles. This particle filter updates particles based on odometry data and then uses the simulated LiDAR scans from the particles to compute the probabilities of the car being at each position. From this, we can find the average location and orientation that we believe the car has.

Once we implement these functions we'll be ready to focus on path-planning. Up to this point all of the labs have focused on specific parts of the car's interaction with the environment, obtaining poses, images, scans, and odometry readings. After this implementation, however, we can now abstract the car into an object who's location, orientation, goal, and surroundings are known. With this abstraction, path-planning will become a much easier task for next lab.

In the following sections we will break down each component, ellaborate on our technical approach and present our findings and modifications that we found to work best. This strategy

## 2 Technical Approach

### 2.1 Motion Model (author: Vlada Petrusenko)

One of the modules of the localization systems is the motion model, which is responsible for updating the position of the car based on the former position, odometry, and noise. Strictly speaking,

$$x_k = [x_k, y_k, \theta_k]^T = f(x_{k-1}, \Delta x)$$

where $x_k$ and $x_{x-1}$ are positions of the robot at times k and k-1, and $\Delta x$ is the odometry data. One of the ways to describe motion is

$$x_k = R \cdot \Delta X + x_{k-1}$$

where R is the rotation matrix that transforms odometry data $\Delta x$ from robot reference frame to the "world" reference frame.
Additionally, we add random Gaussian noise to the theoretically computed new position, so that the particles spread out, so the finalized computation is

$$x_k = R \cdot \Delta X + x_{k-1} + p_{Gaussian}(std, \mu)$$

A more visual representation of the motion model is in the figure below. Every next position of the car is described by the odometry data (dotted arrow on the figure), and respectively position of the car is updated. Noise is not illustrated, but little Gaussian noise is added to the data.

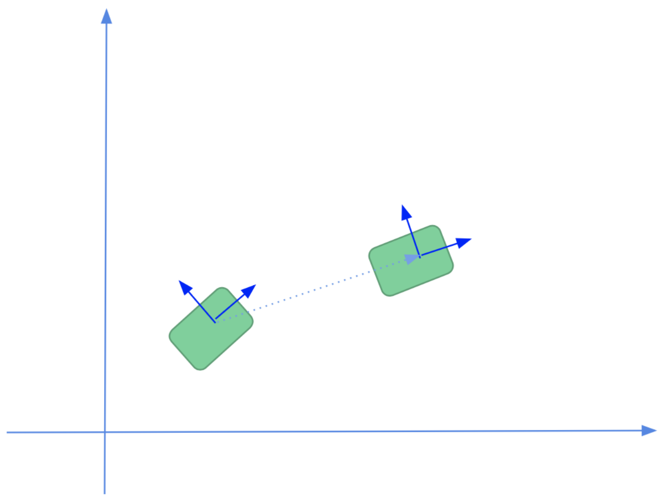


Figure 1: Updating the position of the car based on odometry

## 2.2 Sensor Model (author: Diego Delarue)

A large step of localization is knowing the car's location. In this case, the sensor model computes the probability of getting the results we are getting assuming that at time $k$ we are at location $x_k$ in map $m$, $p(z_k^{(i)}|x_k, m)$. In order to try and predict this probability, we need to break apart the problem into different probabilities as follows:

$p_{hit}$ represents a Gaussian distribution centered around the ground truth distance. This is what makes the most sense and is the driving probability component to calculate the total probability $p(z_k^{(i)}|x_k, m)$.

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

$p_{short}$ represents the probability of sensing an object that is in front of an farther expected object.

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

$p_{max}$ handles reflected measurements that cause a spike in the maximum range by discounting the weight that they impact the probabilities.

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

$p_{rand}$ represents unforeseen effects of sensing in the real world.

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

From these definitions we can come up with constants that add up to one $\alpha_{hit}, \alpha_{short}, \alpha_{max}, \alpha_{rand}$. The initial parameters were set as follows:

$$\begin{aligned} \alpha_{hit} &= 0.74 \\ \alpha_{short} &= 0.07 \\ \alpha_{max} &= 0.07 \\ \alpha_{rand} &= 0.12 \end{aligned}$$

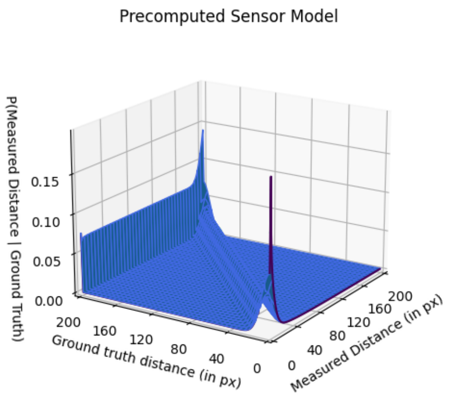
$$p(z_k^{(i)}|x_k, m) = \alpha_{hit} * p_hit + \alpha_{short} * p_+ \alpha_{max} * p_{max} + \alpha_{rand} * p_{rand}$$

In order to discretize these probability functions, however, we decided to set $\epsilon$ to 1.0 so it represents a single pixel in the lookup table. This way we have:

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

With these chosen functions and a chosen table width of 200, we can move on to the precomputation of the sensor model table, $T$. The reason why we decided to precompute a table of size 200x200 with $z_k^{(i)}$ and $d$ as its axes was in order to save time. The value of $\eta$ for $p_{hit}$ is determined in order to normalize $p_{hit}$ across each column $d$. Instead of computing these probabilities that require involved calculations, we can create a 2d array that will look as follows:

$$T[z_k^{(i)}][d] = p(z_k^{(i)}|x_k, m)$$



Once these values are precomputed, whenever we receive new scans we can evaluate them and return the corresponding probability. However, before using $T$ we preprocess the particles as well as the ground truth with the following steps:

Initially defining $squash = \frac{1}{2.2}$,

1. scale particle scans and ground truth by map resolution and map scale

2. clip particle scans and ground truth $x$ to $\min(\max(0, x), 200)$
3. Downsample particle scans and ground truth to ‘num_beams_per_particle‘
4. obtain probabilities $p$ of all $N$ scans by obtaining the corresponding values of $T[z_k^{(i)}][d]$
5. squash probabilities by squashing parameter $squash$, $p = p^{squash}$

The squashing was a preemptive strategy to make the probability distribution less peaked.

It is as this point that a matrix of size $N$x‘num_beams_per_particle‘ is output for the particle filter to use.

## 2.3 Particle Filter (author: Jason Lee)

We use a particle filter in our implementation of Monte Carlo localization to infer a distribution of pose estimates for the robot car. Each particle is a vector representation of the car's pose, the format of which matches what's used in the motion model. The distribution of particles is updated whenever we receive odometry or LIDAR data. These updates are done in parallel through the use of ROS callbacks, which allows us to operate at the different update frequencies of the motion and sensor models rather than bottlenecking one of the models. However, this necessitates a threadsafe approach to handling the particles. Therefore, we utilize a lock on the array of particles that is acquired before any of the callback functions modify the array, which is then released afterwards.

### 2.3.1 Odometry Data Update

Our particle filter is subscribed to an odometry topic that provides us with the most recent odometry data of the car containing the linear $x$ and $y$ velocities, as well as the angular $z$ velocity. This is then passed into the motion model to apply the frameshift as well as some noise to the particles.

### 2.3.2 LIDAR Data Update

Our particle filter is also subscribed to a LIDAR sensor topic that provides us with its most recent set of readings. This is then passed into the sensor model to generate a list of weights representing whether or not each particle is consistent with the sensor readings. A new set of particles is then re-sampled over the valid pose estimates.

### 2.3.3 Pose Estimate Calculation

The pose estimate the particle filter publishes is an ”average pose” of all the particles' positions. We considered two different approaches to calculating this average.

Firstly, the approach we settled on was a circular mean of all particles.

$$x_{mean} = \frac{1}{n}\sum_{k=1}^{n} x_k$$

$$y_{mean} = \frac{1}{n}\sum_{k=1}^{n} y_k$$

$$\theta_{mean} = \tan^{-1}\left(\frac{\sum_{k=1}^{n} \sin\theta_k}{\sum_{k=1}^{n} \cos\theta_k}\right)$$

While this approach was satisfactory for the performance of our car in simulation, we acknowledge that this is not suitable for multi-modal particle sets, which occur very frequently near the beginning of Monte Carlo localization before the position of the car converges.

To remedy this weakness, we considered an approach to distribute each particles' values into bins (similar to those of histograms) and generate the pose estimate from the most populous bin. While this would force the most frequently appearing bin to be chosen in the case of multi-modal data, there were other unresolved issues regarding bin size and the representative value for each bin that we did not have time to experiment with.

## 3 Experimental Evaluation (author: Vlada Petrusenko)

Our model was tested and evaluated on the simulation, and on-car testing is one of the future potential evaluations. Particularly, our model was expected to follow a specific route.

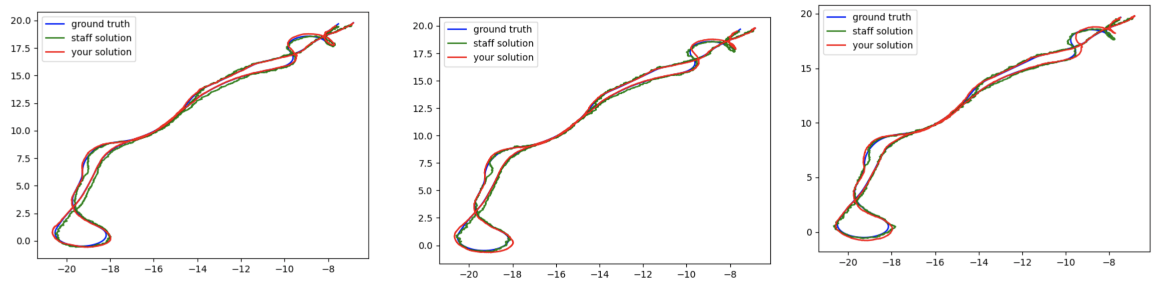


Figure 3: Simulated functionality of our model vs reference ground truth at different levels of the odometry noise.

While working on optimizing the parameters and get the best possible performance, we had to experiment with the noise in the motion model, probability components, and squash in the sensor model.

The most optimal noise for the noise model was with a standard deviation of 0.01 (so that it is enough to deviate from the path, but not enough to disrupt the system.

A hypothesis we had in terms of the failing cases when first testing was that squashing the probability distribution to not have peaks at all could be leading to confusion when translating the scans that were being taken into the car's possible odometry. This came as a result of us noticing that there were already no significant peaks in our probability distributions and saw no need to squash them further as well as the car clearly having no awareness of its location or orientation. This hypothesis turned out to be right as changing the parameter from $\frac{1}{2.2}$ to 1.0 resulting in significantly better performances.

In terms of sensor probability components, they appeared to be quite well-tuned initially, and any retuning of those resulted in worse results than the original.

## 4 Conclusion

Our work on the localization project went through theoretical calculations, simulation of the result, and in the future will include implementation on the physical car for path following. There are also a number of areas we would like to further improve the design of our localization system. Firstly, as mentioned previously we would like to experiment with a pose estimate calculation that is more robust to multi-modal distributions that occur in the early stages of Monte Carlo localization. Additionally, during simulation we achieve a publishing rate of 18.78 Hz. To further improve on this to reach a 20 Hz publishing rate, we plan on running a profiler to determine which module is the largest cause of bottleneck and how to optimize it.

## 5 Lessons Learned

- Diego: I learned about how localization works and it is tricky to piece apart into modules since it is all truly integrated in the particle filter. Working around busy schedules as the semester ramps up was also increasingly a challenge, especially with spring break in the middle. Communication and planning, however, became key as we had to work around each other's schedules as well as set deadlines for each component of the lab.

- Vlada: I learned that some of the challenges of this class (and any other big project) are not purely technical, but also logistical. Particularly, these project weeks were very tricky about scheduling and coordinating work, which got us off track for some time. One of the most valuable lessons from it was realizing the situation, communicating our situation with the staff members and each other, and constructing an action plan for the future. From the technical side, I had a learning experience in debugging a complex system on Gradescope, without having a very detailed idea about the bugs and being limited by the test run time.

- Jason: I learned about the importance of parallelizing our workflow for the labs - the motion model and sensor model were obvious starting points, but while the particle filter required both to be completed for evaluation, it could have been worked on simultaneously. Additionally, it was extremely helpful to have another set of eyes on the code that I worked on to help diagnose potential issues and bugs, and accepting the help progressed the lab much more than working on my assigned section alone.
- Jack: I gained much more experience on debugging ROS nodes and subscriber/publisher relations, especially as it related to how they differ between simulation/autograder to the physical car. Additionally, I learned a lot about optimizations and tradeoffs in this lab as update frequency has potential real-world consequences, and there was a balance to be found between modularity of the particle filter and potential function overhead.