Lab 5: Monte Carlo Localization

Team 20

April 8, 2019

1 Overview and Motivations

Author: Rowan Cheung

In this lab, we seek to give our robot the ability to estimate its location and bearing in a real world map. The robot's location is approximated as the weighted average of a set of particles that are tracked with the robot's movement. The weights are determined by the likelihood of sensing the observed LIDAR data from the position of that particle. Thus, the resulting location estimate is odometry dead reckoning honed by sensor readings. Through the use of this probabilistic particle filter, we develop a localization algorithm that is robust to noise and sets a foundation for navigation.

2 Proposed Approach

2.1 Motion Model

Author: Christopher Hughes

The motion model is very basic. It subscribes to odometry that is published by the racecar at a rate of 50 Hz. The subscription yields velocities in the x, y, and θ (rotation around z-axis) directions. The model translates these velocities into a ΔX_B by multiplying by the change in time since the last odometry message (1/50 sec). This $(3, 1)$ vector of odometry data is then applied to the particles following a rotation matrix as shown in Figure 1. This rotation matrix is applied to the odometry using a numpy function: $np.einsum('ijk, jl- > ki', rot, odom)$. This function multiplies the size $(3, 3, n)$ rotation matrix by the $(3, 1)$ odometry matrix along the second axis (**j-axis**) and then transposes the output so the resulting matrix is $(n, 3)$ where n is the number of particles. This is exactly the shape of the particles to begin with so now we can just add the two together and we get the updated particles with odometry applied.

$$rot = \begin{bmatrix} np.cos(th) & np.sin(th) & np.zeroes(n) \\ -np.sin(th) & np.cos(th) & np.zeroes(n) \\ np.zeroes(n) & np.zeroes(n) & np.ones(n) \end{bmatrix}$$

Figure 1: Rotation matrix to be applied to odometry data (n is the number of particles, th is the third column of the particle matrix that has a point for every row; the angle)

However, if we were to just simply apply the odometry our car would actually be off from reality due to real life noise from wheel misalignment, wheel sliding, and inexact power to velocity conversions. Thus, we want to account for this noise in an efficient and effective manner. This was done by applying Gaussian noise to every-point with a standard deviation of .008. This may seem low, but for our implementation of the combined algorithms it ends up working out.

2.2 Sensor Model

Author: Eric Chen

As the motion model uses odometry changes to progressively expand the reach of the particle distribution, the sensor model prunes unlikely particles in favor of particles with higher possibilities. The sensor model is essentially a probability distribution over observations. For each particle position returned by the motion model, a ray tracing algorithm uses the known environment map to generate the laser scan array that would be sensed if the robot had the exact same pose as the particle. The actual LIDAR measurement observed by the robot (z_t) is then used to assign an existence probability to each particle - $p(z_t|x_t, m)$, or the probability that the robot makes the observation it did (z_t) given that it's at particle x_t (with particle LIDAR scan defined as z_t^*) and we know the stationary map m. Given an array of particles and their associated probabilities based on the real LIDAR measurement, we choose a subset of likely particles and guide the spread out particles towards convergence at the robots true location.

The probability distribution used to assess the likeliness of particles is decomposed into a sum of four parts. Each part models a specific measurement scenario. They are defined mathematically as shown in Fig. 2.

The first component (p_{hit}) models the likelihood that the LIDAR beam hits a known object in the environment, at a reasonable distance. This is modeled as a Gaussian distribution centered around the ground truth measurement. If the actual LIDAR measurement is close to the ground truth (z_t^*), the particle probability will be high. The second component (p_{short}) models the likelihood that the LIDAR beam hits an unknown object at a close distance. These detected objects could include unknown non-static objects moving through the environment, or even LIDAR beams reflected off of the robot itself. This probability is

Equations
$p_{hit}(z_t\|x_t,m) = \begin{cases} \eta \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z_t - z_t^*)^2}{2\sigma^2}\right) \text{ ; if } \quad 0 \le z_t \le z_{max} \\ 0 \text{ ; otherwise} \end{cases}$
$p_{short}\left(z_t\|x_t,m\right) = \frac{2}{z_t^*} \begin{cases} 1 - \frac{z_t}{z_t^*} \text{ ; if } \quad 0 \le z_t \le z_t^* \\ 0 \text{ ; otherwise} \end{cases}$
$p_{max}(z_t\|x_t,m) = \begin{cases} 1 \text{ ; if } \quad z_t = z_{max} \\ 0 \text{ ; otherwise} \end{cases}$
$p_{rand}(z_t\|x_t,m) = \begin{cases} \frac{1}{z_{max}} \text{ ; if } \quad 0 \le z_t \le z_{max} \\ 0 \text{ ; otherwise} \end{cases}$

Figure 2: Equations used to calculate each of the four probability components used to obtain the total probability distribution - $p(z_t|x_t,m)$.

Constant	Value
α_{hit}	0.74
α_{short}	0.07
α_{max}	0.07
α_{rand}	0.12
σ	$0.5m$
z_{max}	$10m$

Figure 3: Constant values used to compute $p(z_t|x_t,m)$ for each particle.

defined by a downward sloping line dependent on z_t^* and z_t. From the equation, we can see that if a particle has a large z_t^* but a small z_t is recorded, this z_t has high probability of being a measurement generated by the close unknown objects described above. The third component (p_{max}) describes the likelihood of detecting a very large measurement. This can occur if the LIDAR beam reflects at a strange angle from an object and never makes it back to the LIDAR puck. This component is modeled as a large spike in probability at a specified z_{max}. This is so that potentially erroneous large measurements don't contribute unfavorably towards a particles probability score. The final component (p_{rand}) simply models randomness in the LIDAR measurement. It is simply introduced as a small constant value dependent on z_{max}. These four components are each weighted by a specific scalar, and summed to produce a single probability for each particle. The weights for each component, as well as the other constants used to compute $p(z_t|x_t,m)$ are displayed in Fig. 3.

In practice, the probability distribution $p(z_t|x_t,m)$ is pre-computed offline during initialization for a discrete set of values. This allows for faster online performance, without sacrificing much accuracy as the distribution itself is an approximation in the first place. The discrete values of z_t^* and z_t used to pre-compute $p(z_t|x_t,m)$ are $[0.01m, 20.01m]$ with stepsize $0.01m$. Each LIDAR measurement made by the physical robot is compared element-wise to ground

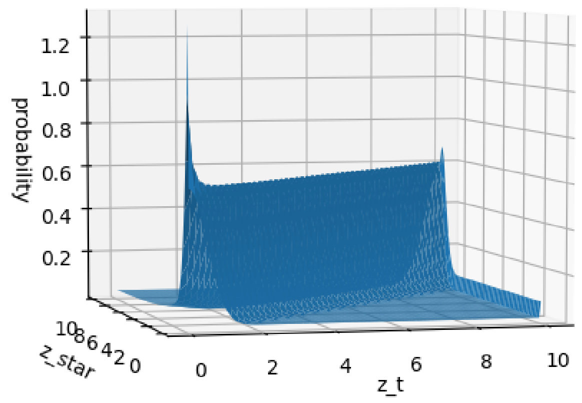

Figure 4: Resulting distribution for $p(z_t|x_t, m)$ where z_{star} is equivalent to x_t

truth scans produced by the ray tracing algorithm for each particle. A probability is extracted for each z_t and z_t^* in each of the two scans. In order to produce a single probability for each particle, the probabilities computed for each laser beam are averaged. Each particle is assigned this average probability, and an array of particles with associated likelihoods is returned.

2.3 Particle Filter

Author: Christopher Hughes

To utilize the full extent of the sensors on the racecar we apply both the motion and sensor updates to estimate our location in the known map. This is done by a variation of the Monte Carlo Localization (MCL) algorithm. This algorithm takes in some initial particles, applies a motion update to those particles, applies a sensor update and repeats the process. Each time it goes through this, the estimation for the location of the robot gets more accurate.

The first step to the MCL algorithm is initializing the points in a semi-uniformly random manner around a best guess of where the robot is in the map. To achieve this, we subscribe to the initial pose publisher in *rviz*. Our algorithm then initializes particles with a Gaussian standard deviation of 0.3

around what the estimated pose was. Once we have the initial particles, we are now ready to apply the motion and sensor updates.

We wanted our MCL algorithm to be robust and to never miss a motion update since we thought the most valuable information was having a reliable update as to where we were going at all times. Thus, for every callback to odometry, we add the message to a double ended queue, or deque, in python. This allows us to easily keep track of our odometry data and never miss a single one.

Since we should not edit the same list of particles by two separate processes, we gated the two functions so one can not run while the other is updating. Our algorithm then alternates between updating all the motions on the odometry stack to updating the sensor messages. This overall produces fairly reliable results and allows us to do all 50 odometry updates and anywhere between 8 and 25 sensor updates, depending on how many particles we initialize our robot with.

3 Experimental Evaluation

Author: Rowan Cheung

For our evaluation, we use a combination of qualitative and quantitative metrics. Our localization is first visualized and inspected by eye. We also calculate our error from ground truth in simulation.

3.1 Manual Inspection

We visualize several values as our robot navigates the Stata basement in both simulation and real-life. First, we show each of the particles in the global (map) coordinate frame. We also depict the calculated average pose as an arrow starting at the average x and y location and angled by the average theta. As our robot moves through the map, we show a trail history of past locations. Figure 5 shows an example of such a visualization in the Stata basement map.

We would've liked to have been able to also visualize the LIDAR scan from our robot, transformed to map coordinates. Unfortunately, we had trouble successfully transforming this data. As we use the same pose depicted by the arrow above to calculate the transform (this pose was determined by eye to be accurate to the robot's global location), we are unsure of the explanation for our transform's poor performance. While the LIDAR data would have improved our visualization, these visual tools were still extremely useful in comparing our perception of the robot's global location with the robot's estimation.

3.2 Error Metrics

As we have no ground truth in our real-world data, we can only compute error in simulation. We assume the simulation odometry data can be integrated over

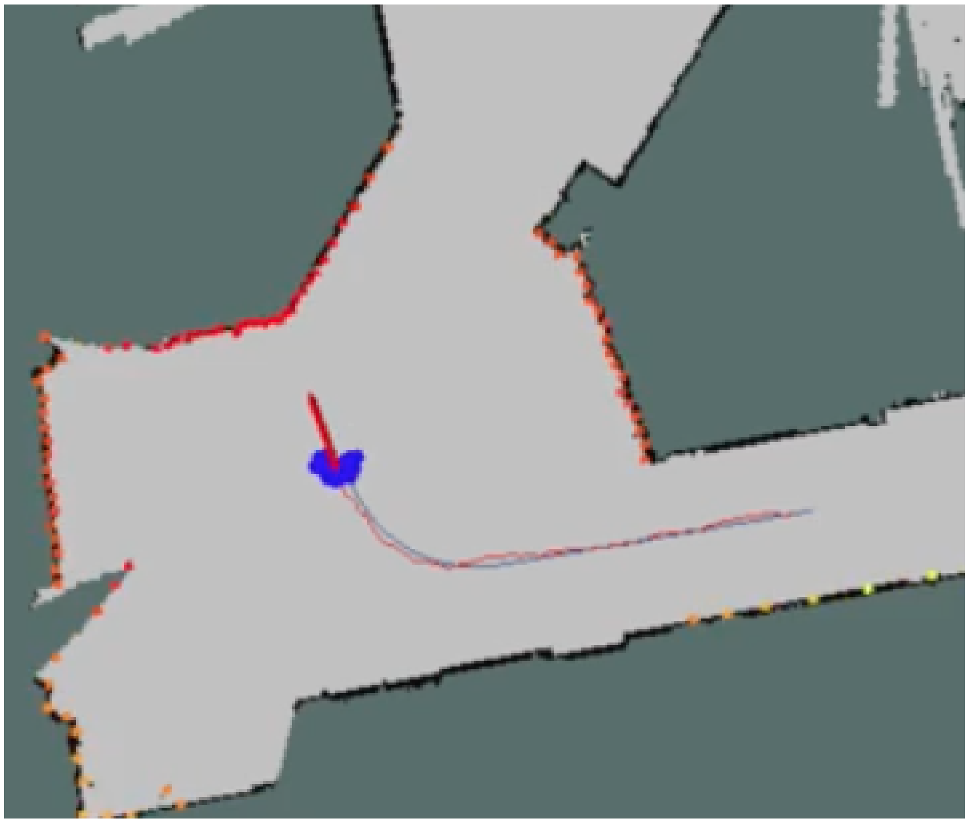

Figure 5: Example of visualization in simulation, depicting particles, average pose, past locations, and LIDAR data

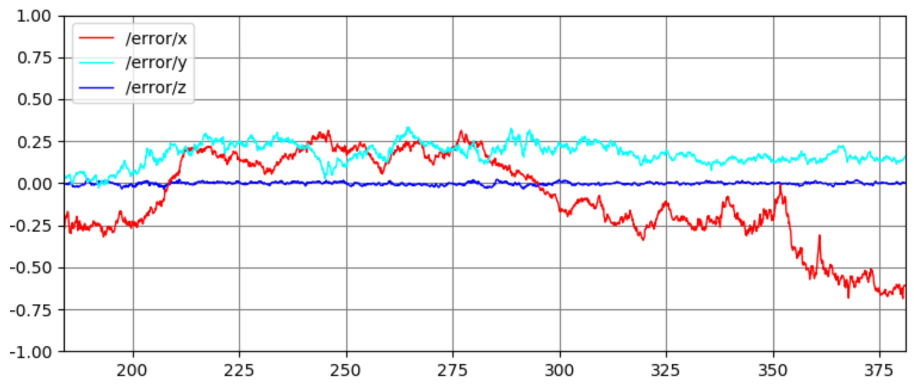

Figure 6: Deviation from ground truth pose coordinates over time in the map frame.

time to produce a ground truth pose result, given the lack of noise in simulation. We then compute the difference in this pose with our pose estimate across time. Figure 6 shows the error over time during a simulation run. We are able to keep error within 0.75 meters over a period of just over 6 minutes. While our pose is maintained quite well, our x and y values deviate from ground truth over long periods of time. This is to be expected as more noise would accumulate over time and could possibly be further improved by further tuning our sensor model.

4 Lessons Learned

Author: Christopher Hughes

Throughout this lab the largest struggle we endured was run speed. Our sensor model was slow, inefficient and was not unusable for localization. However, with effective implementation of all of our updates using numpy matrix operations, we were able to dramatically speed up the calculation and actually produce reliable fast speeds. While speed was a huge problem in this lab and we optimized our code a lot, there is still a lot of room to further speed up and improve our code so that it runs faster and is able to do more of the sensor updates.

One of the hardest problems we dealt with on this lab is issues with translation from working in simulation to working with the car. The maps of Stata basement are not entirely accurate and is abundantly noisy, which makes it harder to localize properly. The noise of motion was different than what was expected; this led us to experimenting with different kinds of noise and eventually settling with a Gaussian distribution with a fairly small standard deviation.

Editor: Carlos Cuevas