CS8803: STR, Spring 2016.
Lab 1: Robot Localization Report

Bhavishya Mittal, Sumant Hanumante, Sai Teja Duthuluri
`(bmittal6, shanumante3, dst3)` @gatech.edu

February 24, 2016

1 Introduction

This lab assignment problem is based on robot localization and particle filtering. Given the layout of a map along with odometry and laser finder readings from a robot, the aim is to accurately find the location of the moving robot in the map.

2 Approach

The basic approach of localizing a robot in the map (map already given) is through using particle filters.

1. Generate a cache of the ray-casting data on map for future use: In this cache, the map is divided into grid cells and for each unoccupied grid cell, the distance to the nearest obstacle is calculated along angles 10 degrees apart in the range 0 to 360 degrees.
2. Initialize a sensor model: This involves caching a matrix which has the dimensions of *max_sensor_range* x *max_sensor_range* and contains the probability values for actual sensor value, given the expected sensor value. These values are computed using p_{hit}, p_{short}, p_{max} and p_{rand} PDFs.
3. Randomly sample particles over all unoccupied locations in the map pointing in random directions.
4. Read in the odometry and laser finder readings from the log file. We consider only the laser sensor readings and take odometer data from it. To avoid taking multiple readings at same location, we take the next laser reading only if the robot has moved considerably (atleast 2dm) compared to previous reading.

5. Run the particle filter algorihm.
6. For every reading, we take each particle and sample from the motion model to get the new particle position.
7. We have the sensor values from the reading and we get the expected sensor values for this newly sample particle from the already cached ray-casted data. Run the sensor model on this to get the weight for each newly sampled particle.
8. Resample the particles based on their weight.
9. Repeat steps $(5 - 8)$ for all log file readings.

3 Implementation

3.1 Motion Model

For our motion model, we consider the *Odometry Motion Model* that uses odometry data as given in textbook. And since we are using particle filters, we use the sampling approach to model the motion of the robot. We repeatedly sample (max_retries = 20) for a new particle until it lies in an unoccupied region of the map. If we do not get such a particle, we explicitly set the weight to 0. This motion model has a number of tunable parameters which have been discussed in a section below.

3.2 Sensor Model

For our sensor model, we follow the basic algorithm for beam models of range finders, as given in textbook. Here, our model is an aggregation of four PDFs: p_{hit}, p_{short}, p_{max} and p_{rand}. Each of these four PDFs have a corresponding weight, speicifed by z_{hit}, z_{short}, z_{max} and z_{rand}. Given a pose reading, we calculate the expected odometry reading using ray casting method. This is done for every particle. The tuning of the z parameters is discussed in a section below.

3.3 Resampling Procedure

We resample particles based on their weights. We first normalize the weights and then align them sequentially from 0 to 1. Then, we sample a number in range 0 to 1 from a uniform distribution and select the particle corresponding to the weight where this number falls. This way, we resample all the particles again. Some modification is done to this resample step to solve robot kidnap problem. This modified version is explained in extra credit section.

4 Parameter Tuning

4.1 Motion Model

The motion model consists of four parameters: α_1, α_2, α_3 and α_4.

The parameters α_1 and α_2 are used to calculate the difference in rotation angle between the estimated and actual robot pose. We do not want this difference to be more than $5\,^\circ$. Based on this limit, we tuned the two parameters so that this difference would always be less than $5\,^\circ$. We have $\alpha_1 = \alpha_2 = 0.005$.

We perform the same procedure for tuning α_3 and α_4, which are used to calculate the difference in translation between estimated and actual robot pose. We want the difference to be less than 1dm in this case. After tuning, the best results were obtained for $\alpha_3 = \alpha_4 = 0.12$.

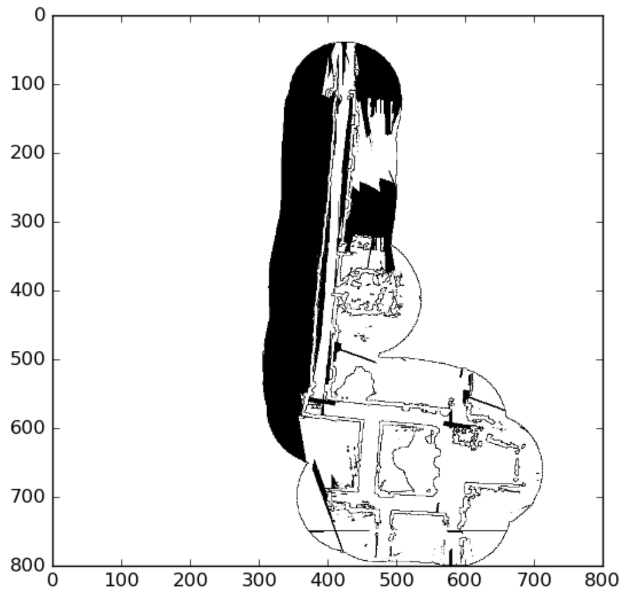
4.2 Sensor Model

The sensor model consists of six parameters. Four for the weight of prabability functions : z_{hit}, z_{short}, z_{max} and z_{rand} and two σ and λ for normal and exponential distribution repectively. The 4 weight parameters were tuned by looking at the PDF curve and observing the corresponding behaviour in the actual map. We set $\sigma = 2$ as we assume that expected sensor reading and observed reading should not be more than 2dm apart. λ was set empirically. Some of the PDF curves have been presented in the Results section below.

5 Results

5.1 Map Model

Given below is a image of the map generated from the cached ray tracing data. We used this to validate our sensor model is working as expected.



5.2 Sensor Model

Given below are a few plots of the sensor models used in our code.

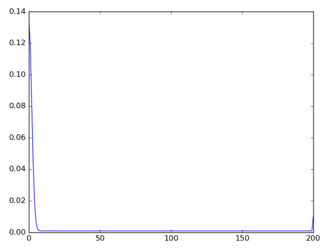


Figure 1: Mean 0

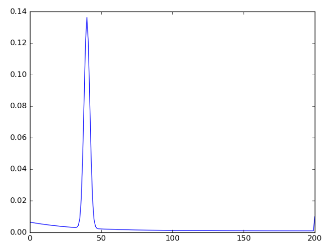


Figure 2: Mean 40

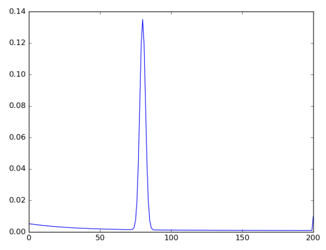


Figure 3: Mean 80

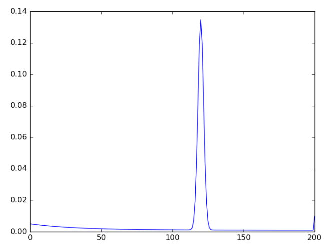


Figure 4: Mean 120

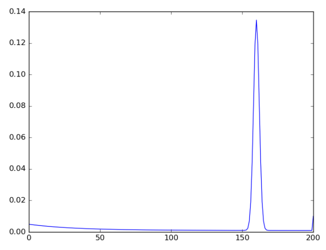


Figure 5: Mean 160

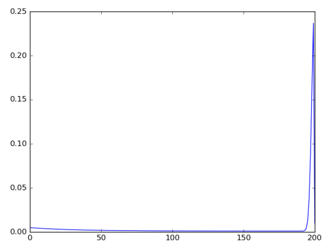


Figure 6: Mean 200

6 Extra Credit

6.1 Kidnapped Robot

For extra credit, we have implemented a robust version of the particle filter which we call **doctoring**. This technique helps the localizaton algorithm by considering alternative hypothesis, and latch on to a better hypothesis, if one exists. We resample the particles in two special cases:

1. The total number of unique particles in the system goes below 20% of the original number of particles.
2. We maintain a randomly generated time epoch: in the range 25-45. When this epoch is reached, if the variance along x and y axis is very low (ie the particles have completely localized), we resample and generate new particles.

The new set of random particles (*num_particles*) generated are given a weight of 1. The already existing particles are assigned a weight of 3. Now we select *num_particles* from this pool. Thus, the already existing particles are three times more likely to be selected. At this point, all the weights of all particles are reset and the particle filter restarts.

An example of the use of this algorithm can be seen in the submitted video file vid_merged_14.mov. Here, the particle filtering algorithm has been run on a log obtained by fusing log1 and log4. At 1:12 mark, the robot turns into a room which signals the end of log1. At this point, log4 begins causing the number of particles in the filter to drop below 20% based on weights assigned by the sensor model. The **doctoring algorithm** kicks in and we can observe the robot being localized at its new position immediately after.

6.2 Adaptive number of particles

In order to analyze the effect of number of particles, we've created a video (vid_varying_particles.mov) showing the different localizations based on the number of particles. The number of particles is shown at the top. From the video, it can be seen that after 5000 particles, the algorithm always localizes to the same starting position. To balance correctness and speed of the algorithm, we chose 8000 particles in our final filtering algorithm.