Project 3: Odometry Based Mapping

Lars Kouwenhoven (lk545)

Introduction

This document describes the approach and results for Project 3 of Intelligent Autonomous Systems, focusing on wheel-odometry, Lidar-based mapping and SLAM.

Problem formulation

For project 3, we were provided with three sets of files, for three different scenarios. The first type of file included wheel encoder information, the second included IMU data, and the third lidar readings. For the first portion of the project, we focused on the encoder and lidar files. The data from these files originated from a skid-steer robot driving around in an indoor environment. The goals were then as follows: (1) map the trajectory of the robot based on the encoder data and the technical specifications of the device, (2) to map the walls and other objects in the environment, as detected by the lidar readings while the robot was traversing its path and (3) to perform simultaneous localization and mapping (SLAM) using both wheel-based odometry and lidar readings.

Technical approach

SLAM makes use of both odometry mapping and lidar readings. Rather than using basic odometry mapping, we make use of a number of 'particles', that are informed by the odometry readings, but include some degree of noise. This is to correct for e.g. slipping, which takes place when a robot is turning. While ranging through the odometry and lidar readings, we will adapt the weights of the different particles based on the correlation of their lidar readings with the existing map. As such, particles that have a high correlation with the existing map, will be strongly represented. Then, we keep track of the number of *effective particles*. When this number reaches below some threshold, we resample the particles. Each of these steps are discussed in more detail in the sections below.

Step 1: Advance robot using odometry mapping

The odometry-based mapping was performed using the encoder data. Each encoder file contains readings for multiple timesteps. The readings consist of the variables FL (rotations of front left wheel), FR (front right), RL (rear left), RR (rear right) and ts (timestep). This information was used to calculate the movement taken place during each timestep *t*. This relies on the fact that, using the *FR* and *FL* readings, one can determine the angle the robot has moved in, as well as the distance it has travelled. For this purpose, the following formulas are used:

$$\Delta\theta = (e_R - e_L) / w$$
$$\Delta x = (e_L + e_R) / 2 * cos(\theta)$$
$$\Delta y = (e_L + e_R) / 2 * sin(\theta)$$

...where θis the angle of the robot at some time *t*.

Then, for each time *t*, we can determine the change of location of the robot, and map this in a graph. This is also the step at which we introduce some noise to each of the particles. Noise is introduced both at the computation of the angle, as well as the computations of Δx and Δy. The earlier formulas then become:

$$\Delta\theta = (e_R - e_L) / w + noise$$
$$\Delta x = (e_L + e_R) / 2 * cos(\theta) + noise$$
$$\Delta y = (e_L + e_R) / 2 * sin(\theta) + noise$$

Different values for the noise were used, ranging between 0.1 and 0.005, the latter of which is the variance of the angle readings.

Step 2: Update particles weights

Originally, each particle will have equal weight (1/n_particles). In this step, we aim to update these weights. Each particle will now have a new position from step 1. Using each of these positions, as well as the lidar reading at time *t* superimposed on the positions, we look at the correlation between the lidar readings and the map as it is at this time step. I have used a simple metric of correlation, which is equal to the sum of log odds values at each of the lidar wall readings. Therefore, we expect a particle with a mapping that is close to the actual map to have a high correlation value.

Next, we multiply the existing weights with the correlation values. The weights will now no longer sum up to 1. Therefore, we apply a softmax function for smoothing, as follows:

$$e = exp(weights - weights_{max})$$

$$weights_i = e_i / \sum_j e_j$$

All the weights will now sum up to 1 again.

Step 3: Resample particles

We define the number of effective particles as $n_{effective} = (\sum_i w_i)^2 / \sum_i w_i^2$. As the name suggests, this is a metric of how many particles still have a significant weight. When $n_{effective}$ is lower than half of the number of particles, resampling is performed. I have done so by generating an array which contains the cumulative sum of weights at each particle index. Then, for each of the number of particles, a random number between 0 and 1 is drawn (e.g. 0.6). We find the first index where the cumulative sum is higher than 0.6, and add the particle at that index to our new set of particles. With this method, particles that have a high weight have a high chance of being

selected, which is exactly what we want to achieve. Particles may (and likely will) be duplicated, but this is not a problem, as new noise is introduced later.

Step 4: Use lidar data to display environment

With the robot trajectory known, we can now use the lidar readings to map the walls of the location. At each time *t*, we have a lidar reading that consists of a set of angles (in radians), as well as a set of ranges. For each *i*th entry in the angles, the *i*th reading in the ranges set corresponds to the range of the lidar sensor at that particular angle. With this information, we know where walls and other objects are located, according to each particle. We use the particle with the highest weight to map the lidar readings at each time *t*.

Important to note here is that I have excluded any lidar readings with a range under 0.15 meters, as this would correspond to the lidar sensor 'reading' the robot itself, as well as any ranges over 20 meters. Then, from a range and angle value, we can find the corresponding x and y coordinates according to the following formulas:

$$x = range * cos(\theta_{robot} + \theta_{lidar})$$
$$y = range * sin(\theta_{robot} + \theta_{lidar})$$

The coordinates map the objects. With the Bresenham algorithm, we can then map the empty cells. The value of object cells is increased by 0.9, whereas the value of empty cells is decreased by 0.1. We also limit the ranges of the log odds values: a cell can never have a value higher than 20, or lower than -20.

Results

Map 20

Figure 1 displays the path taken by the robot in scenario 20, based solely on the odometry reading. Figure 2 shows the mapped walls in scenario 20, and figure 3 explicitly shows the empty spaces as well. From figure 2, it becomes clear that the space is not mapped perfectly: in the upper part of the image, we can see that that same wall is displayed twice, each time with a slightly different angle.

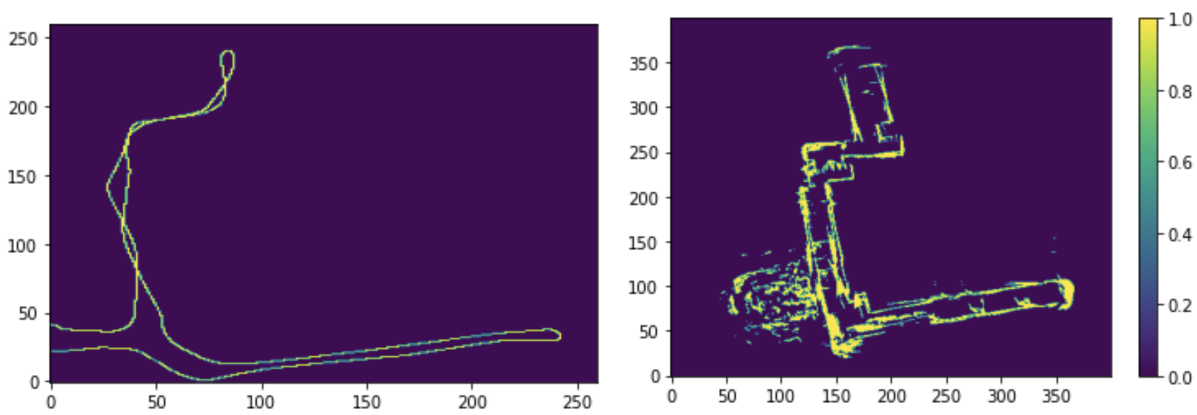


Figure 1 *Figure 2*

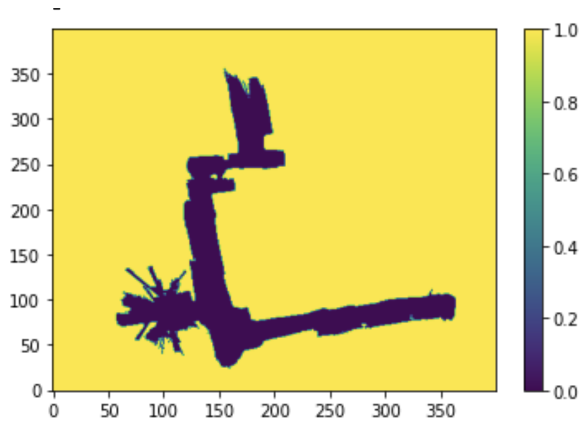


Figure 3

We aim to improve these results with the use of SLAM. The result using 50 particles, and with a noise of 0.005 is shown in *Figure 4*.

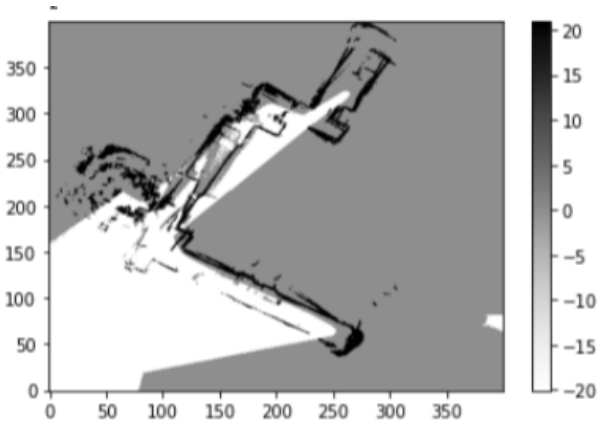
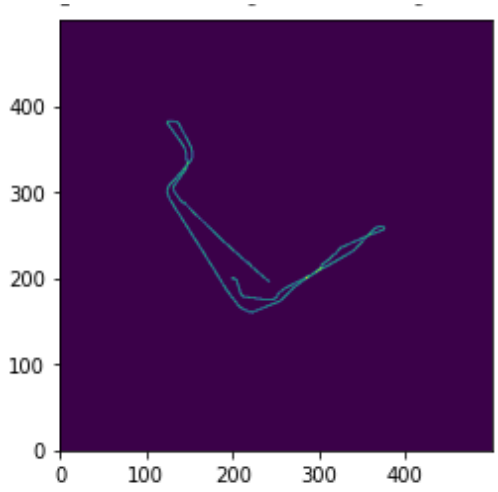


Figure 4

Map 21

Figure 5 shows the result of simple odometry mapping, whereas *Figure 6* displays the results when using SLAM.



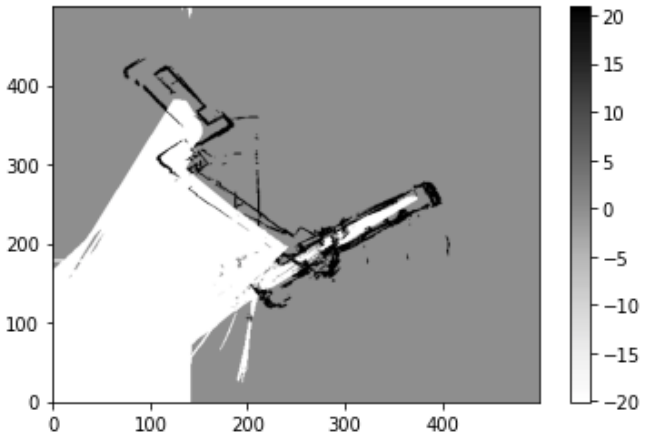
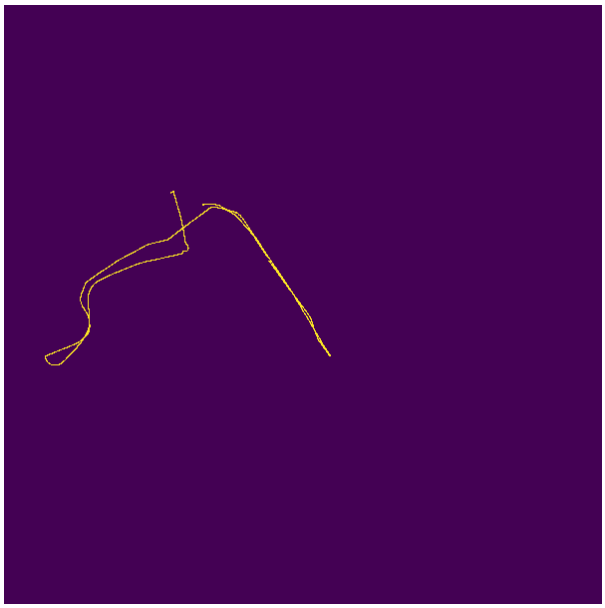


Figure 5 *Figure 6*

Map 22

Figure 7 shows the robot path, whereas *Figure 8* displays the results including lidar readings.



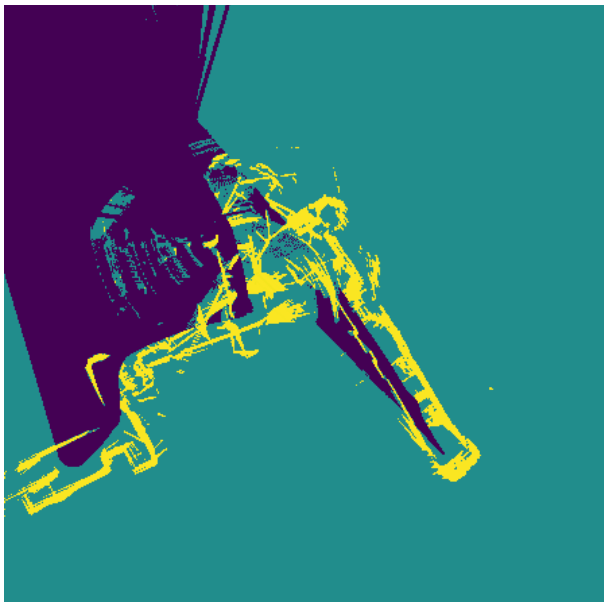
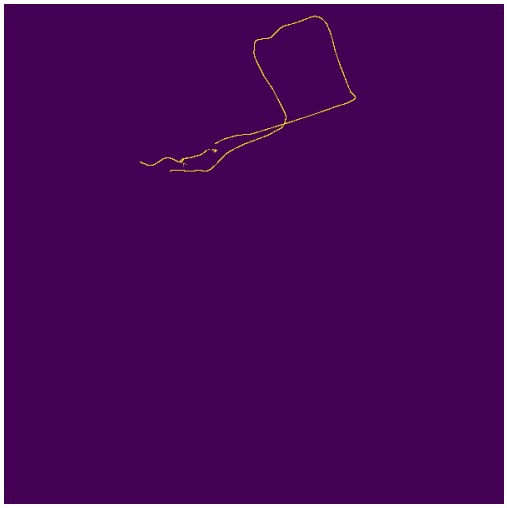


Figure 7 *Figure 8*

Map 23

Figure 9 shows the robot path, whereas *Figure 10* displays the results including lidar readings.



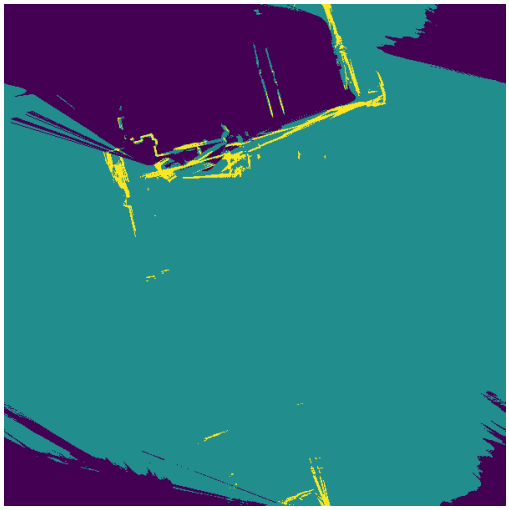
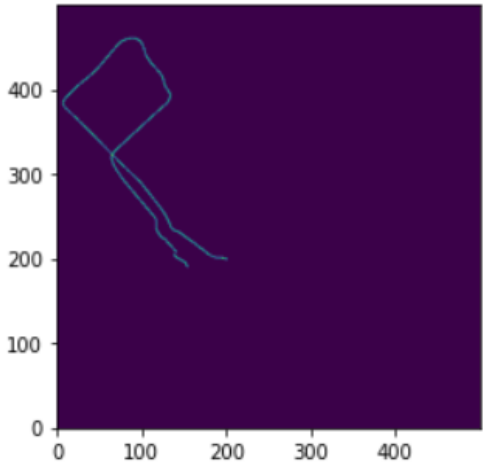


Figure 9 *Figure 10*

Map 24

Figure 11 shows the robot path, whereas *Figure 12* displays the results including lidar readings.



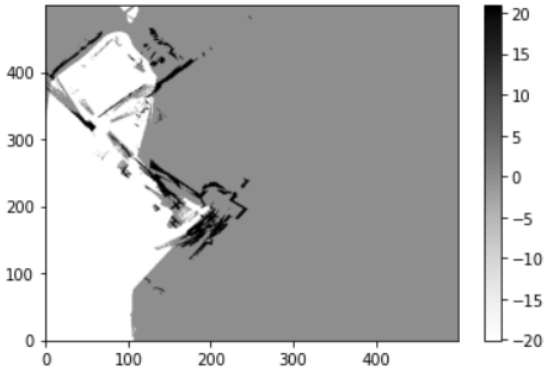


Figure 11 *Figure 12*

Discussion

From the results, it is clear that the SLAM is not entirely functioning the way it should. Rather than removing noise, it appears to introduce noise at certain points. My suspicion is that this has something to do with the way in which I defined my correlation. However, at inspection of different particle readings, the particles that achieve the highest correlation seem to line up with the particles that best represent the situation.

I did also notice that, when I increase the noise (to 0.05), certain particles are selected which deviate strongly from the actual path. These particles have a high weight, and thus completely throw off the path. All subsequent Lidar readings are then displayed incorrectly.