Technische Universität Berlin

Robotics Course

Assignment 4: Localization and Mapping

Group Members:
Changxin He
Paul Molloy
Yang Xu

Student ID:
406129
415274
373470

19.01.2020

1 A Bayes

From the problem description, it is known that,

$$P(z = 42|open) = \frac{2}{3} \tag{1}$$

$$P(z = 42|\neg open) = \frac{1}{3} \tag{2}$$

$$P(open) = \frac{3}{5} \tag{3}$$

The conditional probability could be calculated via prior probability and posterior probability:

$$\begin{aligned} P(open|z = 42) =& \frac{P(open, z = 42)}{P(z = 42)} \\ =& \frac{P(z = 42|open) \cdot P(open)}{P(z = 42|open) \cdot P(open) + P(z = 42|\neg open) \cdot P(\neg open)} \end{aligned} \tag{4}$$

Finally, the probability of the door being open given the measurement z = 42 and our prior knowledge about the door is:

$$P(open|z = 42) = \frac{\frac{2}{3} \times \frac{3}{5}}{\frac{1}{3} \times \frac{2}{5} + \frac{3}{5} \times \frac{2}{3}} = \frac{3}{4} \tag{5}$$

2 B Mapping with Raw Odometry

2.1 Simple Counting Method

Our solution creates a map by using the laser scan sensor data. It works by using the simple counting method. The map is represented by a discretized grid of areas where an obstacle is present or not. There are both a misses and hits 2d double vectors following this representation. The actual occupation map is represented by a 1d double vector of (width x height) length. Each time new sensor data is obtained individual sensor readings of an angle and distance are translated into an x, y position in the map. These points are counted as hits and the position in the hits vector is incremented. Next to get data about misses we suppose that any point between the laser sensor and a point that we hit is probably unoccupied because the laser could pass through it. To do this we calculate the Bresenham line between the robot sensors co-ordinates and the hit co-ordinates. This returns the co-ordinates of discretized map positions that approximate the line the laser is believed to have followed. The corresponding entries in the misses 2d vector are incremented by 1 also. At every time step the occupation map grid position values are update to be hits/(hits + misses). This results in a quite accurate map.

2.2 Results

We verified our results using the provided maze_corridor bag. We saved screenshots shown below and attached, which show the constructed map at 2, 8 and 20 seconds.

(a) Map at time = 2s

(b) Map at time = 8s

(c) Map at time = 20s

Figure 1: Mapping Progress Over Time

2.3 Comments

We hypothesize that the map differs from reality due to noise in the laser sensor measurements. If the robot moved more slowly through the maze then there would be more hit miss data-points for each grid position and so the map would approach reality further.

3 C Monte Carlo Localization: Particle Filter

3.1 Uniform Particle Distribution

The uniform particle filter was implemented in *initParticlesRandom* by populating the x, y and theta parameters of each particle with values generated using the provided uniformRandom utility function. This generates particles which have parameters which are randomly distributed through the possible state space with equal probability. The weights were all assigned to the filter with values $1/n$ where n is the number of particles.

3.2 Gaussian Particle Distribution

The uniform particle filter was implemented in *initParticlesRandom* by populating the x, y and theta parameters of each particle with values generated using the provided gaussianRandom utility function. This generates particles which are distributed with normal noise away from the given mean. The provided mean and standard deviation parameters for each state parameters were used to do this. The weights were all assigned to the filter with values $1/n$ where n is the number of particles.

3.3 Sample Motion Model Odometry

We implemented the odometry motion model sampling in *sampleMotionModelOdometry*, it works by using the measurements from the odometry sensors in the motors but assuming that the odometry values do not match reality. There is some noise between the odometers delta and the real delta. First We get the translation delta and the two rotational deltas based on the difference between the given old and new x, y and theta values.

Then for each particle in the particle set we create a possible actual delta rotation 1, 2 and translation by adding independent but identically distributed Gaussian noise for each particle. Then

for each of these particles we apply its newly generated noisy delta translation, rotation 1 and rotation 2 to the particles last belief of state parameters x, y and theta. Then we assign these new parameters x, y and theta back into the particle for the next iteration.

3.4 Likely-hood Field Range Finder Model

Next the likely-hood field range finder model was implemented. This was done by for each particle going through the lasers scans. Then computing where we think the laser hit in the map grid co-ordinates based on where the particle believes the robot is and the laser distance and angle relative to the robot.

Then if the position is sensible i.e. its in the map and not farther than the max range the particle weight is increased by the likely hood maps positions value at the position of the believed hit co-ordinates.

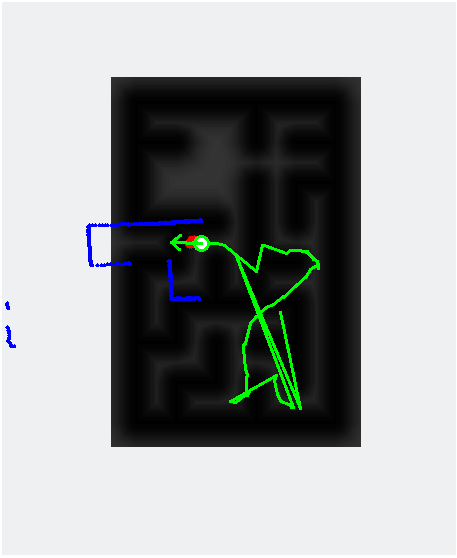


Figure 2: Likely-hood Field

3.5 Re-sampling Step

The re-sampling step is used sample a new set of particles from the current set taking into account their current weights aiming to represent particles with higher weights in the new sample more than particles with low weights. This is done visually by assigning each particle a portion of the perimeter of a circle proportional to the percentage of the the sum of the weights that its weight makes up. A random float is then generated less than the weight of the first particle. This is used as a starting point from the zero mark on the 'wheel'. This is the first sample to be used as the first particle in the new particle filter. The position is then repeatedly increased by *1/sumOfWeights* to get the next sample for the particle filter.

This is implemented in the code by creating a cumulative distribution function array where each element contains the sum of the previous weights and itself. A random float is generated as a starting point for the currentPosition. Then in a loop position is then repeatedly increased by *1/sumOfWeights* and each time a particle is sampled to be added to the new particle filter based on

the particle filter with index equal to the first CDF array value greater than the current position. In this way the new particle filter particles are sampled in a way with the same characteristics as with the sampling wheel illustration.

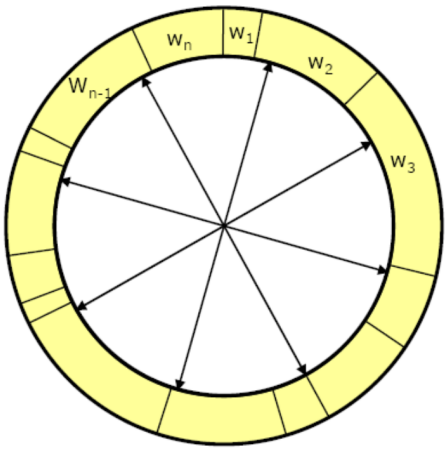


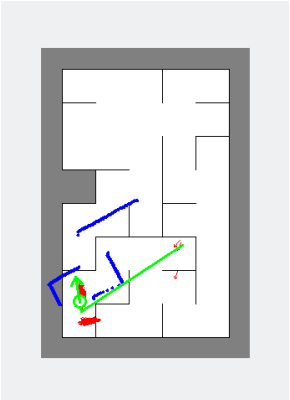
Figure 3: Re-sampling Wheel Source: Robotics Slides

The best hypothesis is then selected by simply looping through all of the particles and finding the one with the highest weight and returning that one.

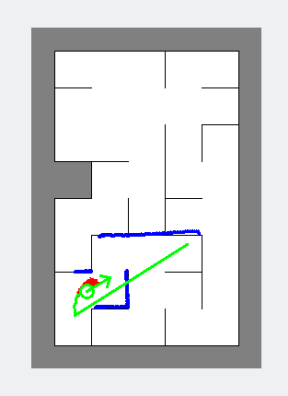
3.6 Testing Results

We tested global localization by tapping the distribute uniform tab once we had started the localization ROS launch file. This generated a normal distribution of red particles throughout the the map. See below the attached screenshots at times 5s, 10s, 14s, 20s, 30s, 42s, 60s and 75s (Note some of the figures are from different runs as it was not possible to get and save all of the screenshots in one go).

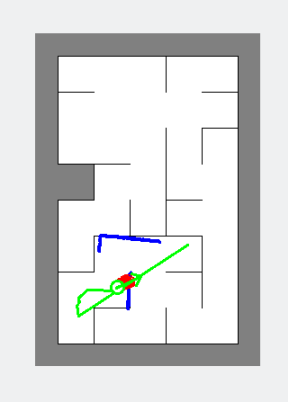
We tested Gaussian localizing by right clicking in the bottom left corner of the map at the beginning and selecting distribute normal.



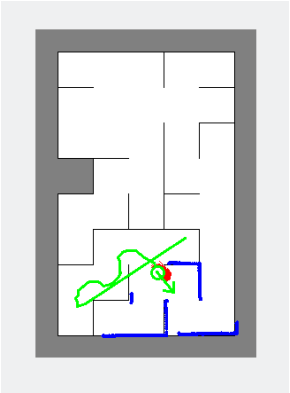
(a) Localization at time = 5s



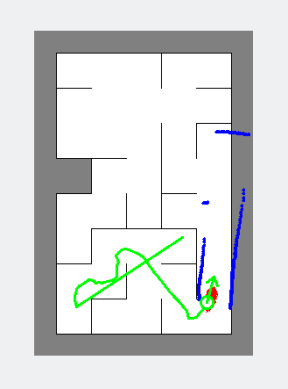
(b) Localization at time = 10s



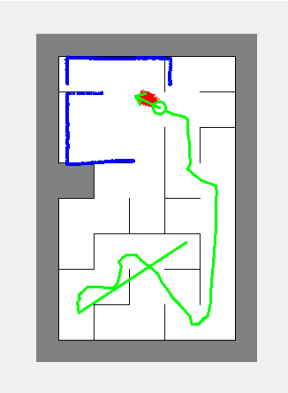
(c) Localization at time = 14s



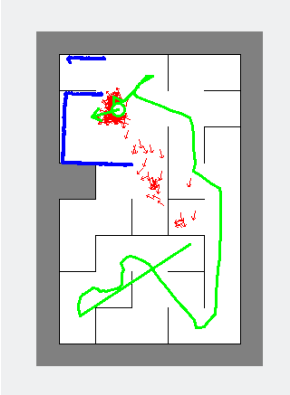
(d) Localization at time = 20s



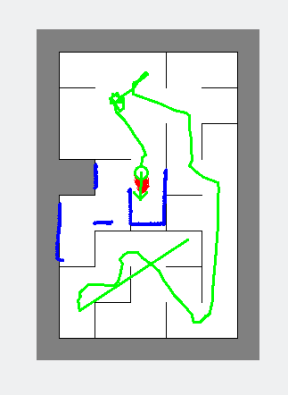
(e) Localization at time = 30s



(f) Localization at time = 42s

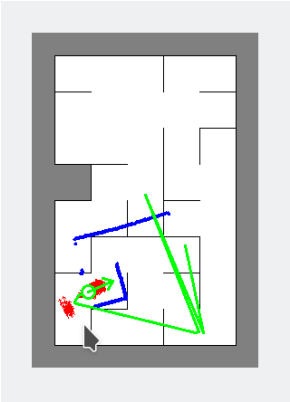


(g) Localization at time = 60s

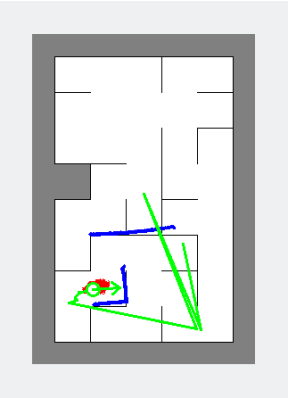


(h) Localization at time = 75s

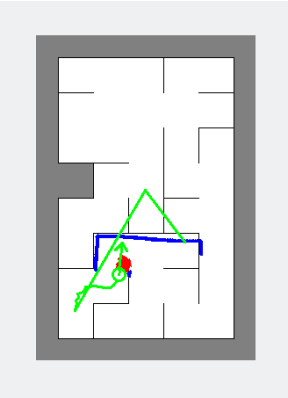
Figure 4: Global Uniform Localization Progress Over Time



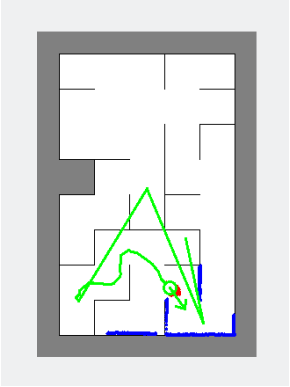
(a) Localization at time = 5s



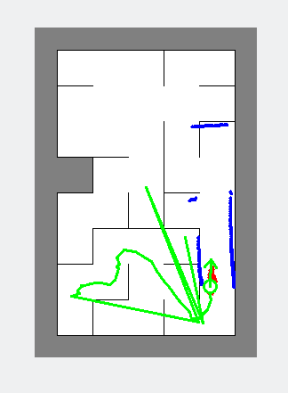
(b) Localization at time = 10s



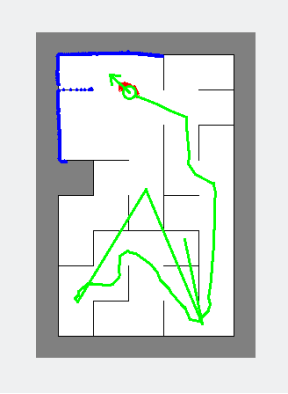
(c) Localization at time = 14s



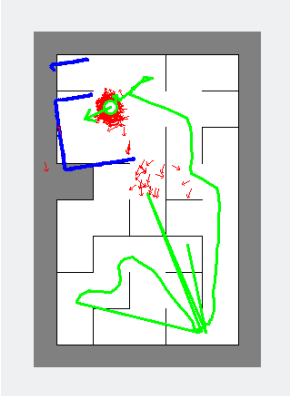
(d) Localization at time = 20s



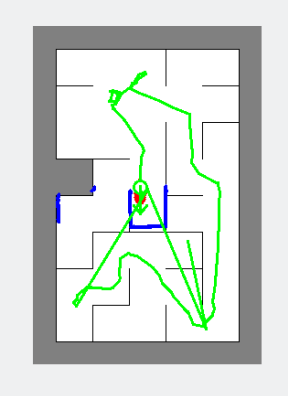
(e) Localization at time = 30s



(f) Localization at time = 42s



(g) Localization at time = 60s



(h) Localization at time = 75s

Figure 5: Gaussian Localization Progress Over Time

4 Task Implementation Table

Student Name	A	B1	B2	B3	C1	C2	C3	C4	C5	C6
Changxin He	33	33	33	33	33	33	33	33	33	33
Paul Molloy	33	33	33	33	33	33	33	33	33	33
Yang Xu	33	33	33	33	33	33	33	33	33	33