Intelligent Robotics - Assignment 2 Report

Team Leonard

November 8, 2019

Thomas Murphy, Alexis Corney, Rehaan Cheema, Esha Dasgupta, Joseph Maliszewski

1 Introduction

The aim of the project was to program a Pioneer Robot to automatically explore its environment and construct a map, used to implement Monte Carlo Localisation using a particle filter. This will allow the robot to localise itself. Environments are by nature unpredictable, due to sensor noise and dynamic objects. Using probabilistic models allows us to overcome these challenges.

2 Method

2.1 Autonomous Exploration

In order for the robot to autonomously explore as much of the environment as possible, it needed to either, have a notion of what areas are unexplored or follow a simple heuristic. Since the area of the lower ground floor that we were to explore was effectively a closed loop, we decided to implement a wall-following program. In order to achieve this, the robot needs to be able to find and follow a wall and stop when it has completed a loop of the environment.

2.1.1 Wall Following

To implement wall following, we split the laser range readings into subsections and took their average. The subsections corresponded to areas in relation to the robots position, from far left to far right. If the robot was detecting obstacles in these sections, we could determine if it was in one of the following states and carry out corresponding actions:

- **Attempting to find a wall** - If the readings from the front-left and front-right fell above a detection threshold it indicated no obstacles or walls. The robot should move forward and veer to the left to ensure that it would eventually return to a wall for it to follow.
- **Found an obstacle** - If the readings directly in front or front-right of the robot were within our detection threshold, the robot should execute a right hand turn so that it no longer faces the obstacle. The robot turns right rather than left as it follows a wall on its left hand side, if it were to face directly at a wall, turning right would place the wall to left hand side to allow it to be followed.
- **Following a wall** - This state is entered if the robot recognises an obstacle to its far left or front left. Once within this state, it travels parallel to the wall. Using a threshold, if the robot determines that it is too close to the wall, it will correct itself through slight right turns until it is a minimum threshold distance from the wall. If the robot exceeds a maximum threshold distance, through turning to avoid an obstacle, it will fall into the 'find-a-wall' state which will veer it back towards the wall. This is how the robot navigates around corners. The threshold values are dynamically decided based upon if there is anything present along the robot's right hand side.

2.1.2 Stopping Exploration

An additional challenge to exploration is getting the robot to stop when when the entire area has been traversed. To do this, we made use of the robot's odometry readings. The program saves the x and y odometry coordinates (we do not need to consider orientation) at the start of its exploration. It will stop its exploration when its odometry pose, which is updated throughout the exploration, falls within a boundary close to its starting point. The boundary accounts for odometry noise which could prevent the robot from returning to its exact starting position.

2.2 Localisation

Monte Carlo Localisation uses a map, odemetry and laser scan data to produce an estimate of the robot's position and orientation within its environment. A particle filter is used where each particle represents a hypothesis (likelihood) of the robots pose.

Initially, particles are generated around a starting estimate of the robots pose combined with Gaussian noise to represent uncertainty.

As the robot moves, the particles are translated according to the action model, which uses odemetry data. A sensor model determines the likelihood of each particle by comparing the actual laser scan readings to the laser scan readings it would expect to see, given the predicted state of the robot. This likelihood determines the weight for each particle. Particles are then re-sampled using an incremental step sampling from a cumulative list of weights, so the new particle cloud contains more particles with higher weights and less particles with lower weights. However, to deal with the "kidnapped robot problem", 10% of the time we add a high Gaussian noise to a sampled particle. This keeps the cloud more dispersed, meaning that the robot will still consider some particles outside of the densest cluster and change their weights accordingly, if the robot is re-located near to outlying particles. There is a trade-off between localisation accuracy and kidnapped robot recovery ability.

The particles converge towards the actual position of the robot, where we use Density-Based Spatial Clustering for Applications with Noise (DBSCAN) to segment the particle cloud into clusters. The heuristic used for DBSCAN was Nearest Neighbours. We then compute the estimated pose of the robot by finding the centroid of the largest, densest cluster.

3 Results

3.1 Map of Autonomous Exploration

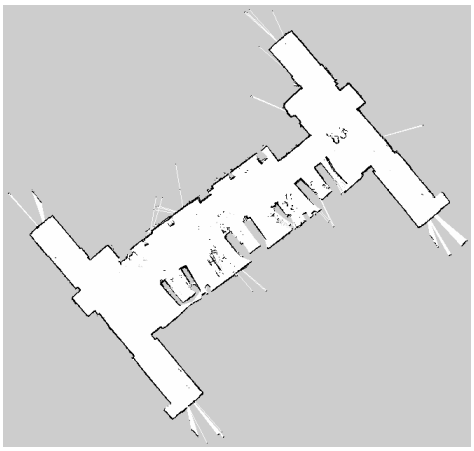


Figure 1: Resultant map from autonomous exploration

Figure 1 shows the map produced after the robot finished its exploration. It successfully mapped the area, although there are some areas of noise, notably the centre of the right hand corridor and around the seating areas. This was caused by people remaining static for long periods of time which led to them acting as obstacles and being labelled as edges by the SLAM algorithm.

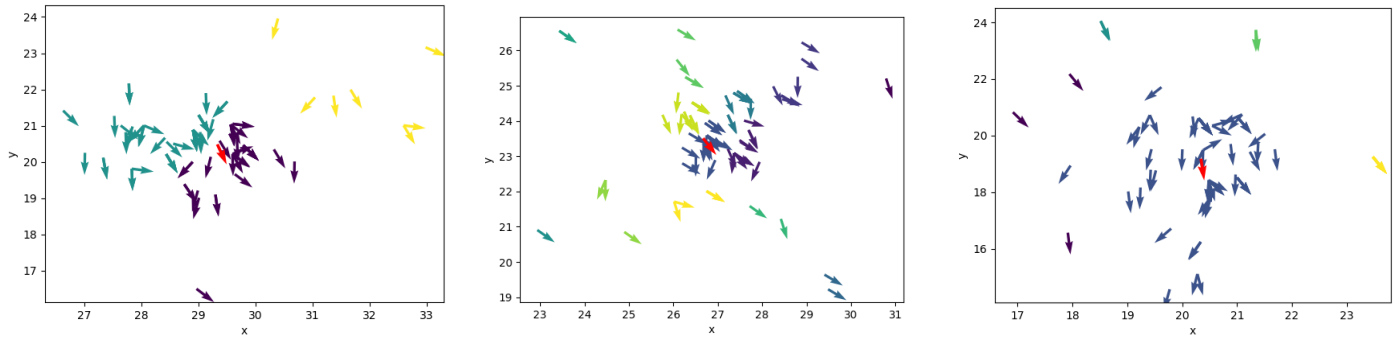
3.2 Determining Parameters for the Particle Filter

Choice of Clustering Algorithm - We tested the K-Means, Affinity Propagation(AP) and DBSCAN algorithms to compare their results. Figure 2 shows a graphical representation of the clusters suggested after 5 iterations of an 80-particle particle filter running with the wall-following code.

Determining the size of particle cloud - The optimum number of particles in the sample was determined to be between 80 and 100. Figure 3(b) plots the error of a range of particle cloud sizes against each other.

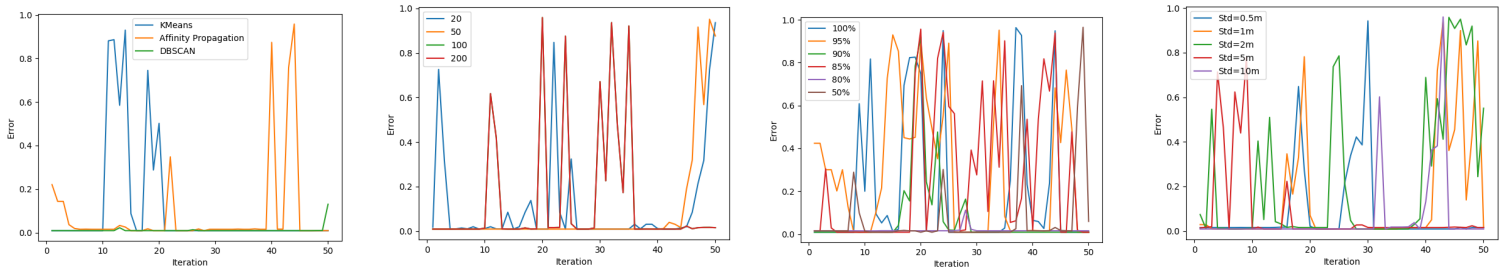
Faithfulness of re-sampling - Figure 3(c) explores the error when varying the proportion of particles which have noise added to them to account for the kidnapped robot problem. The results showed that as the faithfulness of re-sampling decreased, the time it took for the robot to re-localize itself to its true pose also decreased, until 85% where this trend inverted. At 50% it became incapable localising itself.

Determining the dispersion of the Particle Cloud - The dispersion of the particle cloud is determined by the standard deviation of the noise added to a small proportion of the re-sampled particles. Figure 3(d) shows the error when varying this value with a particle cloud of 80 particles and a 90% faithfulness in re-sampling.



(a) K-Means (b) Affinity Propagation (c) DBSCAN

Figure 2: Results of clustering the particle cloud using different clustering algorithms. The estimated pose is marked with a red arrow.



(a) Error in the different clustering methods with 80 particles and 90% faithfulness in re-sampling

(b) Error when varying the size of the particle cloud with DBSCAN and 90% faithfulness in re-sampling

(c) Error when varying the faithfulness of re-sampling the particle cloud with DBSCAN and 80 particles

(d) Error when varying the std of the noise in the particle cloud when updating with the movement model, with 80 particles and 90% faithfulness in re-sampling

Figure 3: Error plots for different experiments after 50 iterations

Faithfulness of re-sampling(%)	Time for localising after kidnapping the robot a distance of 5m (s)
100	≥ 600
95	64
90	22
85	11
80	16
50	cannot localise for kidnapping

Table 1: Table to show how faithfulness of the re-sampling affects the recovery time of a kidnapped robot. Localisation quality deteriorates with quicker recovery from kidnapping.

Std of noise to sampled particles (m)	Max distance robot can be kidnapped to (m)
0.5	2
1	2
2	3
5	8
10	particles too sparse for localisation

Table 2: Table to show how standard deviation of part of the re-sampled noise affects the recovery distance of a kidnapped robot. Done with 80 particles. Localisation quality deteriorates with a greater recoverable distance.

3.3 Analysis: Determination of most effective particle filter model

Choice of Clustering Algorithm: DBSCAN identified the largest, densest cluster most effectively because it is a density based clustering algorithm. Its results are shown in Figure 2(c), the blue arrows depict the main cluster, outliers are shown in different colors. K-Means,2(a), was not able to identify the largest cluster, as it divided the largest intuitive cluster into two. It also required the number of clusters to be specified a priori, which is limiting. Affinity Propagation, 2(b), was also ineffective at identifying the intuitive

main cluster, with it being divided between numerous non-dominant clusters of a similar size. Figure 3(a) plots the error of the three methods against each other, and clearly shows DBSCAN as having the lowest error compared to AP and K-Means. DBSCAN has many advantages: It has a notion of noise, which makes it robust to outliers. Lone particles will not be considered a cluster. The number of clusters is determined by the algorithm diving the data into 'n' dimensions and not by the user. This is useful as the optimum number of clusters cannot be predetermined as the number changes based on the environment. There are also no limitations on cluster shape.

Determining the size of particle cloud: Testing a lower sample size than the optimal range of 80-100 particles resulted in inaccurate readings. Outliers have higher statistical significance in small sample populations, whereas a larger sample diminishes an outlier's significance. A sample that is higher than 150 was shown to be too computationally expensive resulting in visible lag of updating. Subsequently, this meant that the algorithm updated too slowly with respect to the real-time position of the moving robot, resulting in localisation error.

Faithfulness of re-sampling: The optimum re-sampling parameter was found to be 90 percent with a trade-off of the lowest error and the robot being able to localize itself within 22 seconds at a distance of 5 metres, a shown in Table 1.

Determining the dispersion of the particle cloud: The variation in standard deviation of noise led to a trade-off between localisation and the distance the robot could recover from after being kidnapped. A good compromise is a value of 5 metres, as, while it presented the largest radius that the robot could recover from, (8m) as shown in Table 2, it also had the lowest error as the number of iterations increased. Any attempt to cover the whole map with particles, to account for any kidnapping range, will require a higher standard deviation and more particles, the latter of which lead to more lag and a much higher error. A high noise parameter led to confusion as the room is fairly symmetrical with many identical features. The particle filter hypothesises that it is in several similar spots simultaneously with the estimated pose switching between them as it gathers more data.

4 Evaluation

- **Hardware Challenges** The robot's hardware is unreliable. We discovered that the laser readings to the right-hand side were intermittently presenting false min-range values. This affected the exploration program right side. To account for this, we changed the exploration algorithm to follow a wall exclusively to the robot's left. We encountered frequent connection errors, reduced by using duct tape and by bypassing the faulty MultiUSB-hub.
- **Edge Cases Autonomous Driving** Testing of the wall-following algorithm uncovered some edge cases that needed to be accounted for.
  - Incorrect laser readings indicated the presence of non-existent objects, which caused the robot to turn unexpectedly. These readings always presented below the minimum range (impossible readings). These unique signatures allowed us to filter them out in the exploration code. We hypothesise that these anomalies were due to emissions from the medical imaging lab as the robot was directly facing this whenever the error occurred.
  - The robot crashed into objects which had low and wide platformed legs, too low for the laser to detect. As this was fundamentally a hardware limitation of the laser, this edge case was handled by using a dynamic berth. A default wide berth would be used to avoid such tricky objects, however if the robot detected obstacles to its top-right, it would swap to a narrow berth to avoid doubling back on the path taken when encountering a narrow corridor.
- **Localisation** There was uncertainty in localisation depending upon the environment structure and accounting for the trade-off between recovering for kidnapping and accuracy for localisation. By attempting to mitigate this with more particles and a higher dispersion, this lead to expensive and slow computation with the robot not localising correctly.

5 Conclusion

Our task was to get the robot to create a map of its environment, independent of any direction from the user, and localise the robot as it moves. Using a wall-following algorithm, the robot was able to autonomously produce a map. A particle filter and Monte Carlo Localisation allowed it to successfully localise while using the wall-following program to navigate. The localisation of the robot within the environment was successful as was able to overcome the kidnapped robot problem, in most cases. Our parameters for the particle filter were chosen to minimise error in localisation, while keeping a sensible range for recovery should the robot be kidnapped. Future improvements could include; replacing wall-following with frontier exploration, and extensions to the MCL algorithm, such as AMCL (Adaptive Monte-Carlo Localisation). Finally, there is uncertainty due to the homogeneous quality of the environment, which could be mitigated by the use of a camera and fiducial markers to mark the areas as distinct from each other.