Intelligent Robotics: Assignment 1

Group 10

October 2022

1 Introduction

Particle filters use particle sets to represent a probability distribution by extracting random state particles from the posterior probability. Essentially, in a localization problem, it is used to estimate the state of a dynamic system by using previous states and a random noise generator to create a map of particles in the actor's physical space and estimate the location of the actor. New particles are calculated from previous particles using odometry with some random noise variation, random particles are added to stop the kidnapped robot problem and then filtered for future states using some distance sensor to represent the nearby physical world.

Our code for this assignment can be viewed here: https://github.com/uob-ir-g10/pf_localisation.

2 Design Choices

2.1 Outlier filtering & Clustering

2.1.1 Initial attempt

Our initial attempt at clustering was to take the mean position and orientation of all our particles and choose 50% of all particles that best minimise a distance and orientation factor. From this new list of particles we recalculate the mean position and orientation and return that as our true localization.

We chose this initial solution mainly due to the simplicity, and to use as a baseline result to help us work out efficiency advantages / disadvantages as a relation to a simple algorithm like ours. A main disadvantage of this simple algorithm was that it can be easily affected by extreme values, which is a problem when there are a lot of random particles floating around, which were introduced later on to deal with the kidnapped robot problem.

2.1.2 DBSCAN

For a more complex approach to our clustering, we wanted a clustering algorithm that both returned particles in their cluster but also removed outliers

rather than trying to assign every single particle into a nearby cluster. We wanted to remove outliers completely as we are working with the average mean so we wanted tight clusters with large numbers of particles in it to give us solid calculations. This brought us to an algorithm called DBSCAN which ticked both of the boxes we were looking for.

Density-based spatial clustering of applications with noise (DBSCAN) is a density-based clustering algorithm, grouping together points with many nearby neighbours and discarding outliers. The main parameters available to us for DBSCAN are ϵ and `min_samples`. ϵ corresponds to the maximum distance between two samples for one to be considered as in the neighborhood of the other - it is therefore crucial to choose an appropriate value relevant to the application. We chose its value to be 2 * `SAMPLE_TRANSLATION_NOISE` since the noise value is the standard deviation σ we're using as a parameter to the Gaussian noise added to sample particles. Since roughly 95% of the area under a Gaussian curve lie within 2 σ away from the mean, setting it to this value gives us a reasonable parameter for our problem. `min_samples` corresponds to the number of samples in a neighbourhood to be considered as a core point. It's clear this should be related to the number of particles running in the particle filter, so it was set to be 10% of the amount of particles on the map, in order to strike a balance between being able to detect the cluster and limiting the amount of smaller, false positive clusters that may arise during localisation.

2.1.3 Position Estimation

Our solution to multi-hypothesis tracking is to take the first cluster returned by the DBSCAN algorithm. Our reasoning is as follows: the ideal situation is one in which DBSCAN returns but a single cluster, indicating that there's only one highly probable location for the robot. If our parameter choices are sound, this should happen most times the clustering is run when the particle filter has successfully localised the robot, since there will be only one cloud of particles, with a few outliers scattered around. The other two situations are that DBSCAN returns no clusters, or returns multiple ones.

When DBSCAN returns no clusters, we can safely assume that we have lost the robot (perhaps as a result of kidnapping) and it is therefore trivial where we should place the estimated pose. Choosing a random particle out of the list, therefore, is not unreasonable.

When multiple clusters are returned, we're in a situation where we have multiple hypotheses that are being tracked. Once again, given our chosen parameters, there is no strong reason to choose one hypothesis over the other, so just choosing any cluster as our position estimation will suffice until the algorithm falls back into a single cohesive cloud. Of course, we could improve the stability of situations like these by looking at different features of each cluster such as number of particles or density, but we choose not to do that

since these situations quickly converge anyway, and higher density particle clouds do not necessarily mean the guessed position is any better.

Finally, once we have picked a cluster to represent our estimated pose, we chose to take the mean position and orientation from particles in this cluster to give us the values for this estimate. This means that the estimate will be relatively stable even when the underlying particles are moving around due to the added noise.

2.2 Particle cloud noise

Another design choice that had to be made was the amount of noise added to the particles upon sampling. We chose to consider two noise values - one related to sampling upon initial pose estimation (initial noise) and one for subsequent sampling of weighted particles (sample noise). This way, we could start out with a slightly wider cloud for our initial estimate, and the algorithm could hone in on a more precise location after it has recieved some additional laser data, as long as the initial estimation is reasonable.

2.3 Number of particles

Our implementation contains two distinct types of particles. The first are "standard" particles, which are those generated around the initial pose. When the algorithm has properly localized particles, there will be a cloud of these "standard" particles around the approximate position of the robot. In addition, there are a number of "kidnapped" particles with completely random positions and orientations around the map.

The standard particles allow for accurate localisation of the robot as it moves around the map, while the kidnapped particles allow for localisation even under the "kidnapped robot" problem. This way, when the robot is put into a random position on the map, and a kidnapped particle happens to be in a position/orientation close to the robot's actual position, the higher weight given to this particle will ensure that eventually a particle cloud will form and the robot will be correctly localised once more.

There is of course a big decision to be made in choosing the right amount of particles in each case. There are multiple trade-offs to be made here; both in the total amount of particles and in the ratio between the two types of particles.

In terms of total amount of particles, the trade-off is clear: more particles will ensure a more robust & accurate localisation of the robot, but will require longer calculations, which can be critical in real-life environments.

The ratio of the two types of particles is also important. Having higher amounts of random particles means that the robot will more easily be able to localise itself when kidnapped, as more potential areas for the robot to be in will be explored in a time step. However, this also make the particle filter

more unstable when the particles have already converged into a cohesive cloud. For example, consider the situation in which the robot is stood idle facing an indistinct wall. With large amounts of random particles, the particle cloud may start to split and teleport around to other walls in the map very easily.

2.4 Adaptive number of particles

With the trade-offs discussed in the previous section, it may be interesting to include an adaptive aspect to the implementation at this point. The simple adaptation we decided to implement was to increase the number of random particles when there is no single cluster given by the DBSCAN algorithm. Indeed, when DBSCAN doesn't find any clusters, we can conclude that the particles are pretty much all over the place and that the robot has been kidnapped or is otherwise unable to localise itself. We can therefore try to converge to more promising hypotheses more quickly by increasing the number of random particles at this time.

3 Experimental Analysis

3.1 Location error over time

A script `log_location_error.py` was created in order to analyse the filter by looking at bagfiles and comparing our algorithm's estimated position with the ground truth provided by the simulation. Fig. 1 shows an example where the robot is initially kidnapped, but gets located at round $t = 140$ where we see the distance away from the ground truth stabilise. However, the volatility near the end of the graph suggests that there may be too much noise in the particle cloud, or too many random particles. We were able to use data like this to tweak the final values of our implementation, or compare the performance of our implementation to the amcl library.

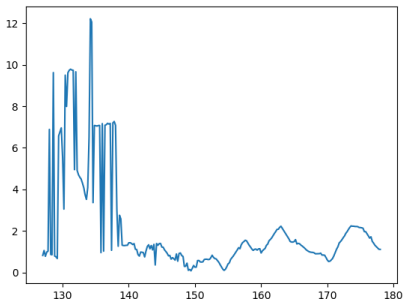

Figure 1: Distance away (m) from ground truth as a function of time

3.2 Sensor Model Prediction

We visualize the sensor model function in Fig. 2 by repeatedly calling the `self.sensor_model.predict(obs, map)` function for different observation values. We can see a sharp drop at the start, due to the model taking into account potential short measurements due to temporary obstacles (such as humans walking in front of the laser). We also see Gaussian curves centered around wall positions indicating the uncertainty of measurements around the physical wall. There is also a non-zero probability everywhere due to randomness in measurements. Finally, we should also expect a narrow spike at the sensor's max range, although it isn't visible in the graph since the max range is set at a single floating-point value, meaning the narrow spike is infinitely small.

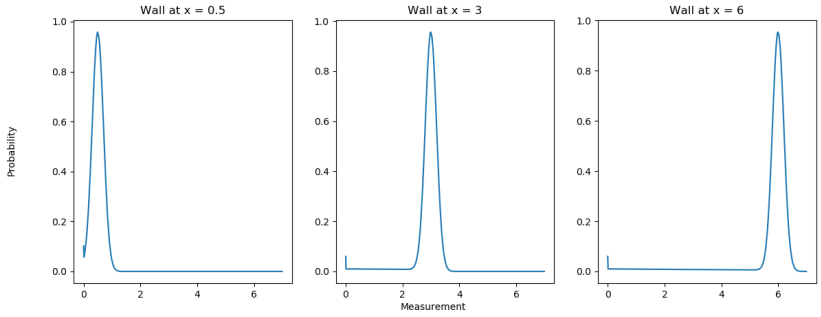

Figure 2: Probability of a given measurement for a wall at $x = 0.5$, 3, 6 m