# Robotics
## Week 5 Practical: Monte Carlo Localisation

Andrew Davison
a.davison@imperial.ac.uk

## 1 Introduction

This week we will follow on from last week's work to complete the components of a full algorithm for Monte Carlo Localisation, a probabilistic localisation filter.

This practical is the last one of three this term that will be ASSESSED. There are 35 marks to be gained for completing the objectives defined for today's practical, out of a total of 100 for the coursework mark for Robotics over the whole term. You should work in your groups on the exercises. Assessment will take place in your groups via short videoconference demonstrations and discussion of your robots and other results with me or one of the lab assistants DURING THE LIVE PRACTICAL SESSION NEXT WEEK, on Friday 25th February.

## 2 Parts of the Algorithm

As explained in lectures, the main parts of one step of MCL are:

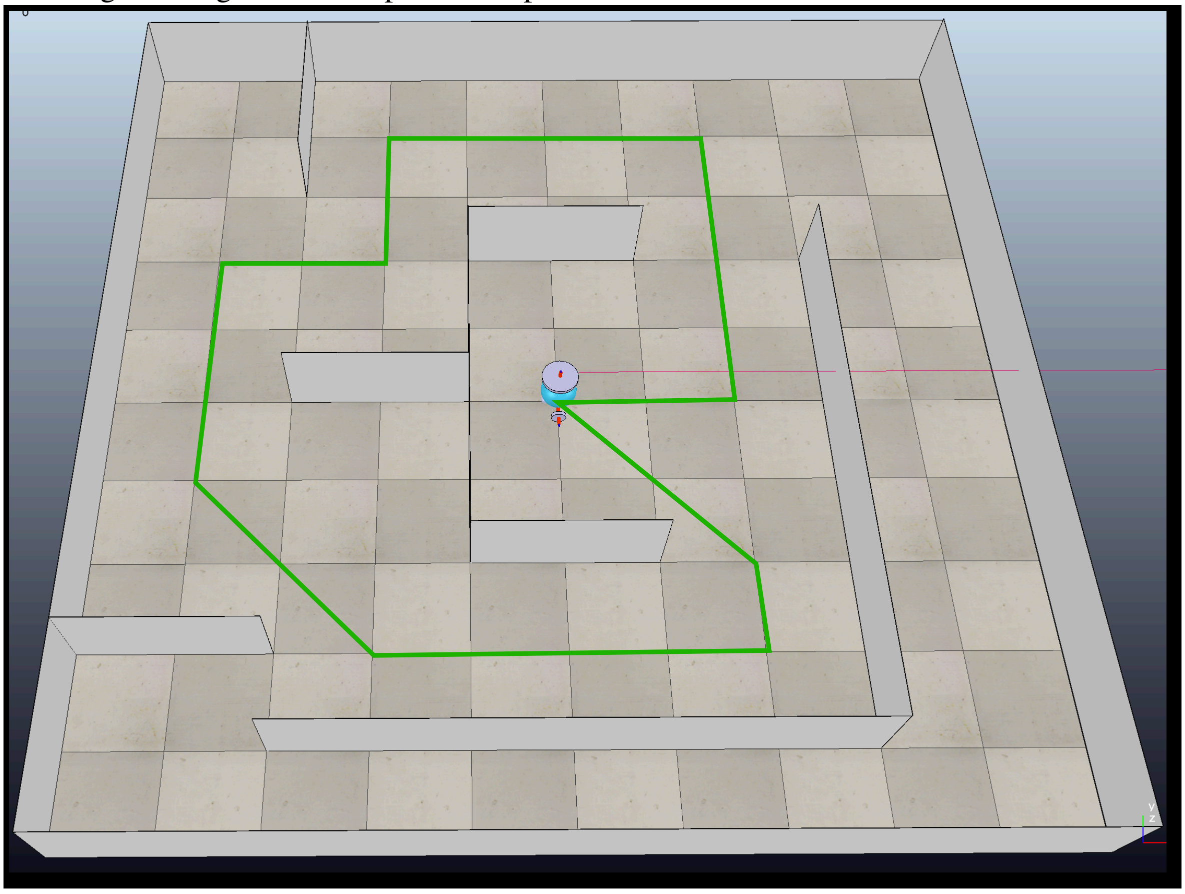
1. Motion Prediction based on Odometry
2. Measurement Update based on Sonar
3. Normalisation
4. Resampling

You should refer to the lecture slides for the details of how to implement these steps. The paper 'Monte Carlo Localization: Efficient Position Estimation for Mobile Robots' by Fox, Burgard, Dellaert and Thrun which is available from the Additional Handouts section of the course website gives more detail on MCL and is well worth reading. We covered step 1 last week. This week you will complete steps 2, 3 and 4. You should continue on from the code you wrote for last week and various things on last week's practical sheet will still be useful.

When the algorithm is complete, your robot should be able to move around the mapped environment in small steps of 50cm, pausing after each movement. During motion, the particle distribution should be 'predicted' to represent the uncertain motion measured by odometry, as we implemented last week. After motion, the robot should make a measurement using its simulated noisy depth sensor (which we will think of as a sonar sensor) of the distance to the wall in front, and use this to adjust the particle weights based on a likelihood function. Then, it should normalise and resample the particle distribution and be ready to complete another motion step. By repeating these steps, hopefully the robot will be able to keep track of its location accurately (at least up to the limits of what is possible with only one sonar sensor).

## 2.1 The Environment and Map

The robots will run in an enclosed area which is defined in the scene file avaiable from:
`https://www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/practical5.ttt`.
The scene has multiple walls as shown here. This picture also shows the waypoint path we want your robot to navigate along in the final part of the practical.



The program script in this scene file is mostly empty apart from two useful things: some code which automatically reads in the configuration of walls from the graphical user interface and saves their end-point coordinates in the array `walls` which we will use in MCL; and an array `waypoints` with the coordinates of the target path we want to make the robot travel through while carrying out MCL in the last part of the practical.

You should cut and paste these useful things from the script as well as the wall configuration from the GUI and combine them with the program you built for Practical 4, because you will be mostly building on that code this week. **Before you do that remember to save a safe copy of your solution to Practical 4 to be able to show us in the Practical 4 assessment.**

## 2.2 Using the Sonar

The depth sensor we will be using like a sonar is the crucial outward-looking sensor which is used to make contact with the mapped world, and we will use the same sensor that you used in practical 3 which is mounted on top of the robot. For the main parts of this practical we will not change the angle of the turret, and keep the sonar pointing forwards.

Because the robot is moving in a large scene, it is reasonable to assume that the sonar will give noisy readings, so find the line in your code where `sensorStandardDeviation` is set and set this to

0.1, to represent a standard deviation of 10cm on all sonar readings in the simulation. You should only make use of this noisy sonar measurement in your MCL program, because a real robot would have to use real sensors.

## 3 Objectives

### 3.1 Sonar Likelihood and Measurement Update (15 Marks)

Based on the information in the lecture notes this week, implement an update function which every time the sonar makes a measurement loops through all of your particles and updates each one's weight by multiplying it by an appropriate likelihood function which corresponds to how well that particle agrees with the sonar reading.

The likelihood function should be implemented as a function
`calculateLikelihood(x, y, theta, z)`
which reads in the $(x, y, \theta)$ position estimate of a particular particle as well as the sonar measurement $z$ and returns a single likelihood value.

This function you define will have several parts. First it needs to find out which wall the sonar beam would hit if the robot is at position $(x, y, \theta)$, and then the expected depth measurement $m$ that should be recorded. The geometry I gave in the lecture should allow you to do this. Then, it needs to look at the difference between $m$ and the actual measurement $z$ and calculate a likelihood value using a Gaussian model as I explained in the lecture. Use standard deviation of 10cm in this Gaussian function to match the noise in the sonar we are using.

Remember that the absolute scale of the likelihood function is unimportant since we will later on be normalising all the particle weights to add up to 1.

**To demonstrate the your likelihood function in the assessment, we will ask you to show us your Lua code for the function on screen and explain how it works.** It is not easy to observe how well the function works by looking at the robot because it will not do much on its own without the other parts of MCL below! But while implementing it you should test that it is working properly by for instance printing out or visualising its calculations.

### 3.2 Normalising and Resampling (12 Marks)

Following the lecture notes, implement the final two steps of MCL which take a particle set whose weights have been adjusted by the measurement step, normalise it so that all weights add up to 1 and then resample it to generate a new particle set.

The normalisation part is straightforward. You need to add up all the weights in the unnormalised set, and then divide the weight of each by this total.

In the genetic algorithm interpretation of MCL, resampling is where strong, successful particles get to reproduce and weak ones die out. Given a set of $N$ particles with varying but normalised weights as the starting point, resampling generates $N$ new particles whose weights are all equal to $1/N$ but whose spatial distribution now represents the previous weighted distribution.

Technically, what happens is that each new particle is a copy (in terms of $(x, y, \theta)$) of one of the old particles. For each new particle, the old particle to copy is selected randomly by sampling from the old set according to their weights. i.e., if one of the old normalised particles had weight 0.2, then a fraction 0.2 of the new particles on average should be copies of this one. Obviously we will get many copies of the strong particles, and few or in many cases no copies of the weak particles.

The random choosing of particles to copy proportionally to their weights can be simply achieved by building a cumulative weight array and then generating random numbers between 0 and 1. i.e. if you have $N$ particles, construct an array where the value of entry $n$ is $\sum_{i=1}^{n} w_i$: the probability of this or any previous particle. Then to make a new copied particle, generate a random number between 0 and 1 and see where it intersects with the values of this array. e.g. if particle $n = 15$ has high weight $w_n = 0.30$, it may cover the range 0.15 to 0.45 in the cumulative probability array. Every time a random number in this range is chosen, a copy is made of this particle. You should use a temporary array space to generate the new particle set, then copy it back to overwrite the main arrays once resampling is completed.

Finally now, you are at the end of the algorithm and can go back to robot motion and the step 1 (motion prediction) of MCL.

You will have to modify the state machine you built for Practical 4 to enable the robot to progress through these different steps.

**To assess this part, we would like to see the whole MCL algorithm running on your robots in the test course. Normally we can assess this section and the next part (waypoint navigation) together.** We want watch the evolution of the particle set in the 3D view and see the correct behaviour, where the particles spread out a little whenever the robot moves, and then tighten up after the measurement and resampling. It will be good if after each moment we have a little bit of time to see what the particles look like after motion prediction, and then how they change after resampling.

An ideal way to prove that your measurement function, normalising and resampling are working during development would be to artificially worsen your odometry model and motion prediction so that on its own it doesn't work well at estimating distance and you get a large particle spread. When moving towards a wall with the sonar and full MCL enabled, however, you should still be able to achieve accurate distance estimates and a well clustered particle set in the movement direction.

## 3.3 Waypoint-Based Navigation While Localising (8 marks)

Last week you implemented a navigation capability for your robot to move to a sequence of pre-programmed waypoints, but we noted that since the robot's position estimate came from taking the mean of particles which were only moving under noisy motion prediction, the navigation would quickly become inaccurate.

Now we should have a better estimate of localisation due to the measurement updates and resampling, so waypoint navigation will become more accurate. Use the same code as last week to make a point estimate of the current position and orientation of the robot by taking the mean of all of the particles.

$$\bar{\mathbf{x}} = \sum_{i=1}^{N} w_i \mathbf{x}_i \ .$$

And then use your waypoint navigation to navigate to waypoints. All of this is the same as last week, except that we should now be getting a much better localisation estimate by taking the mean of our more tightly packed particle distribution.

The lists of waypoints we want the robot to follow is now quite long, and we have included it in the program included in the Practical 5 scene file. You can also see the desired path of the robot in the picture earlier in the sheet. The robot starts at the origin and makes a circuit around the walls.

You will see if you plot these out that a robot that is localising well will not need to do any obstacle avoidance to follow this sequence of landmarks. Just moving in straight lines between them and turning on the spot to point to the next landmark will be enough. Your robots should move and follow this path while performing Monte Carlo Localisation steps at every waypoint.

If you include a ‘repeat’ function in your program, the robot should be able to keep going and follow the waypoint sequence many times.

If it is quite hard to get this all working perfectly; the functions all need to be correct (test them!), and you may need to adjust again a little the motion prediction noise values of your robot to get the particles to spread a suitable amount. But it is very satisfying if it works! And you can see the contrast easily with pure odometry localisation if you comment out your measurement update, normalisation and reampling code — the robot will drift, shown by ever-spreading particles, and will likely hit one of the walls before too long.

Clearly MCL with only a single sonar sensor is quite tricky, because the sensor only measures in one direction and therefore only constrains particle spread in the direction it is facing at the current time. As the robot rotates and looks at different walls, generally the spread of particles can remain constrained overall. Sometimes the robot can make some surprising recoveries of localisation even when it looks like it might be getting lost. It is interesting to work on this quite limited sensing configuration because it really shows the power of probabilistic methods to get the most out of real noisy sensors.

## 4 If You Have More Time

That is the end of the assessed practical, but if you get things all working and are interested in investigating more, a few interesting additional things to try are:

- Make the turret move, and make measurements in different directions as the robot moves. It is quite easy to take account of this in the measurement function; you just need to add the turret angle to $\theta$ for each particle to predict the distance to a wall in a direction which is not forward. In different parts of the scene, different turret angles are more useful, or you could just turn the turret continuously.

- Add extra sonar sensors; for instance you could have sensors looking left, right, forward and back at the same time. You will get four measurements, and just need to multiply all of the likelihoods together to get a weight for a particle. This should make your localisation much better and keep the particles tightly bunched at all times.

- Try global localisation, which is localisation where the start position of the robot is unknown (rather than perfectly known as we have assumed so far). The MCL algorithm is the same, except you need to randomly initialise the particles across the whole range of the environment (and with random angles) to represents no knowledge about the robot’s position. As the robot moves and makes measurements in different directions, poor guesses will be weeded out and the estimate might converge. I think that to get this to work reliably, using more sensors (such as a full ring) will be needed, and also you will probably need more than 100 particles. You’ll also need to think about how to navigate the robot, because the mean estimate of the particles will be meaningless when they are all spread out — perhaps wall following would work.