1 Introduction

This week we will revisit robot motion and sensing but now with a probabilistic viewpoint, understanding how to model and reason about uncertainty. This practical lays the groundwork for next week’s session where these components will be brought together into a full algorithm for Monte Carlo Localisation, a probabilistic localisation filter.

This practical will be ASSESSED. There are 25 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. Assessment will take place via a short demonstration and discussion of your robots and other results to us or our lab assistants AT THE START OF THE PRACTICAL SESSION ON THURSDAY NEXT WEEK, 19th FEBRUARY. No submission of reports or other materials is required. We will assign marks based on our judgement of whether each objective has been successfully achieved, and I will confirm these marks to you every week by email. At the end of term I will create a dummy coursework on CATE which will be used to fill in your total coursework mark.

2 Objectives

2.1 Representing and Displaying Uncertain Motion with a Particle Set (15 Marks)

In Practical 2 we saw clearly that even after careful calibration, a robot’s estimate of its motion based on odometry measurement will always have uncertainty. In today’s lecture I explained how in probabilistic robotics, one way represent this uncertain estimate and how it changes over time is via a particle distribution. The figure below shows a particle distribution evolving over time to represent the growing uncertainty in the position of a robot navigating using only odometry.

Write a program which represents the uncertain motion of your moving robot via a particle distribution to achieve a similar result, and displays the results in the form of a cloud of particles drawn in real-
time using our Pi web interface (see below). The particles should move when the real robot moves, and spread out gradually to represent its growing uncertainty during this motion. No outward looking sensors are to be used in this part.

You will need to demonstrate this program to us next week. Program your robot to follow a 40cm trajectory as in Practical 2 using position control, though make your robot move in 10cm steps, stopping after each movement or turn. In preparation for next week your robot should run on the lab carpet rather than paper, and you may need to alter your tuning calibration parameters slightly to get accurate motion on carpet.

At the start of motion, the set of particles should all be initialised to the origin \((x = 0, y = 0, \theta = 0)\). Each time the robot stops, you should update the position of the particles using the equations below:

- After a straight-line period of motion of distance \(D\):

\[
\begin{pmatrix}
x_{\text{new}} \\
y_{\text{new}} \\
\theta_{\text{new}}
\end{pmatrix}
= \begin{pmatrix}
x + (D + e) \cos \theta \\
y + (D + e) \sin \theta \\
\theta + f
\end{pmatrix}
\]

- After a pure rotation of angle \(\alpha\):

\[
\begin{pmatrix}
x_{\text{new}} \\
y_{\text{new}} \\
\theta_{\text{new}}
\end{pmatrix}
= \begin{pmatrix}
x \\
y \\
\theta + \alpha + g
\end{pmatrix}
\]

Here \(e, f\) and \(g\) are noise terms, which are generated for each particle separately (i.e. different random numbers for each particle so that they spread out) by sampling random numbers with zero mean and an appropriate standard deviation from a Gaussian distribution. Adding a small random amount onto the position of each particle like this will cause the spreading you see in the diagram. Adjust the standard deviations of the Gaussians you sample from to give an amount of spreading which agrees approximately with the amount of uncertainty you think your robot has in its motion. It is best to err on the side of too much uncertainty in the way that the particles spread out so that we can see what they are doing.

The following sections have some more details.

### 2.1.1 Web Interface and Displaying Output

For this and future weeks, we have set up a new web interface to the your Raspberry Pi Robots which makes it much easier to interact with programs running on the Pi from any device on the college network (PC, tablet, etc.). We are providing a web server, implemented in node.js, which you should install on your Raspberry Pis. You can then connect to this webserver from your PC or device. Note that we have also created a new easy web-based way to query your Pi’s current IP address which provides immediate click-through to your Pi’s web interface when you provide the MAC address.

You can copy programs to the Pi through this interface, run them and see text output in the browser. Also, it enables real-time 2D graphics output from your Pis via special formatted Python print statements.

You will still write programs in Python in the usual way, and can still copy them to your Pi and execute them over scp/ssh as before.

For full instructions on how to install the web server, access it from a PC or device, copy and run Python programs and use the special drawing commands, see: [http://www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/webviewer.txt](http://www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/webviewer.txt)
2.1.2 The Particle Set

The set of particles, each of which has values for $x_i = (x_i, y_i, \theta_i)$ and weight $w_i$, can be stored in pre-allocated Python arrays of length $\text{NUMBER\_OF\_PARTICLES}$. A suitable initial value for $\text{NUMBER\_OF\_PARTICLES}$ this week is 100.

In this week’s practical, the weights $w_i$ of the particles will not be important and should just be set to $1/\text{NUMBER\_OF\_PARTICLES}$. The weights become important next week when we incorporate sonar measurements.

The web interface provides a simple way to display a set of points on screen to represent the positions of a particle set. You should experiment to find a suitable scale so that the particle set stays within the bound of the interface window at all times.

2.1.3 Generating Random Numbers

Random numbers for use in the particle movement can be easily generated in Python using the `random` module. See [http://docs.python.org/2/library/random.html](http://docs.python.org/2/library/random.html) for full documentation. In particular, this week you will need to generate random numbers sampled from a Gaussian distribution. The function `random.gauss(mu, sigma)` generates a random number sampled from a Gaussian distribution with mean $\mu$ and standard deviation $\sigma$.

2.2 Waypoint Navigation (5 Marks)

The proof of good localisation is if it can be used to achieve accurate navigation. You can make a point estimate of the current position and orientation of the robot by taking the mean of all of the particles:

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

Based on this estimate you can then control the robot to move towards a target location. Note that this week, this mean estimate is just calculated from odometry so is not very interesting — as the particles spread out as uncertainty increases, their mean will not change — but with this machinery in place you will be set up for next week when the particle distribution will also be affected by sonar measurements.

Provide your robot with the capability to perform position-based navigation via simple path planning through a set of waypoints. A waypoint is an $(W_x, W_y)$ position relative to the world coordinate frame $W$ through which the should pass. Write a function `navigateToWaypoint(float X, float Y)` which, given the robot’s current estimated location $(x, y, \theta)$, drives it to the waypoint at $(W_x, W_y)$. Most straightforwardly this will be achieved by first a turn on the spot through the appropriate angle and then a straight forward motion of the right distance. Refer back to the lecture notes from week 2 on position-based path planning, and please use the same 2D coordinate frame convention defined in the figures in that lecture such that the robot starts at $(x, y, \theta) = (0, 0, 0)$ with its forward direction aligned with the $x$-axis, the $y$ axis points left and positive $\theta$ representing a turn to the left.

Prove the operation of your path planning function to use by demonstrating a short interactive program where a user is asked to enter $(W_x, W_y)$ coordinates in metres which the robot will then move to before asking for more coordinates.

(Again, bear in mind that at this stage your robot is navigating only based on odometry so there will be inevitable drift in its position over long motions; but with this machinery in place we will be ready to
implement a full MCL localisation and navigation system next week.)

2.3 Sonar Investigation (5 Marks)

The sonar is the crucial outward-looking sensor which we will use to make contact with the mapped world in Monte Carlo Localisation next week.

This week we will make some initial quantitative investigation of its properties. We will attempt to calibrate the sonar by comparing the values it returns with ground truth obtained from measurements with a ruler or tape measure (there are some around the lab and I should have some spares). Set up your sonar sensor with a simple program to read continuously and report depth values to the screen.

2.3.1 Sonar Calibration Questions to Answer

On a piece of paper, prepare brief answers to these questions which we will ask you about at the assessment:

1. When placed facing and perpendicular to a smooth surface such as a wall, what are the minimum and maximum depths that the sensor can reliably measure?

2. Move the sonar so that it faces the wall at a non-orthogonal incidence angle. What is the maximum angular deviation from perpendicular to the wall at which it will still give sensible readings?

3. Do your sonar depth measurements have any systematic (non-zero mean) errors? To test this, set up the sensor at a range of hand-measured depths (20cm, 40cm, 60cm, 80cm, 100cm) from a wall and record depth readings. Are they consistently above or below what they should be?

4. What is the accuracy of the sonar sensor and does it depend on depth? At each of two chosen hand-measured depths (40cm and 100cm), make 10 separate depth measurements (each time picking up and replacing the sensor) and record the values. Do you observe the same level of scatter in each case?

5. In a range of general conditions for robot navigation, what fraction of the time do you think your sonar gives garbage readings very far from ground truth?

Acknowledgments

Thanks to Lukas Platinsky, Jan Jachnik, Robert Lukierski, Jacek Zienkiewicz and Duncan White for huge help in preparing the software, hardware and documentation for this practical.