Robotics
Optional Practical: SLAM Simulation in Matlab

Andrew Davison
ajd@doc.ic.ac.uk

1 Introduction

In this (unassessed) practical we will experiment with a simulation of SLAM (Simultaneous Localisation and Mapping) using the mathematics and engineering application Matlab.

The SLAM simulation was developed by J. Neira and J. D. Tardós of the University of Zaragoza, Spain. The scenario is a mobile robot with wheel odometry and a laser range-finder sensor which is driven around a square corridor. There are point features around the corridor whose positions relative to the robot can be observed by the sensor. The features are mapped and their locations estimated along with that of the robot within an Extended Kalman Filter (EKF).

2 Setup

Boot into Linux and download this file from the web:
`http://www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/SLAMLab.tar.gz`
Unpack the file and run Matlab with the commands:

```
tar xvfz SLAMLab.tar.gz
cd 'SLAM Data Association Lab'
matlab
```

Matlab is essentially an interpreted programming language with much powerful mathematical functionality built-in. The main program of the SLAM simulation is the text file `slam.m` in the `SLAM Data Association Lab/` directory. This is a program in the Matlab M-file format. Other files in the `tools/` directory contain functions called by the main program.

3 Starting the Simulation

In the main Matlab window, type `slam` to start the simulation. This starts the main function defined in `slam.m`. A window marked 'Figure 1' will pop up with a view of the simulation scenario: a set of red points represent the ground-truth locations of landmark features which the robot is able to observe as it moves around a square, corridor-like trajectory. The simulation will be paused and pressing Enter will then take it through the Action Sequence below, pausing again after each action. At any time when the program is paused you can hit `Control-C` to come out (then type `slam` to start again).

4 Displays

Alongside the main Matlab window, various displays will pop up which have the following roles:

Figure 1	Ground Truth	Shows the ground-truth layout of features
Figure 2	SLAM Map	Shows the latest estimated robot and feature positions with uncertainty ellipses (in blue) on top of the ground truth positions (in red)
Figure 3	Observations	Shows the latest sensor measurements and uncertainties in these (in green) in the robot's coordinate frame
Figure 6	Compatibility	Shows how feature measurements are checked for matching using different methods
Figure 7	Matching	Shows the assigments by different methods of which observations match which features

5 Action Sequence

Each time you hit Enter, the simulation will carry out the next action of the following sequence (note that at the very start of the simulation, there are extra 'Get observations' and 'Update' steps before the robot moves for the first time to stock the map with some initial features):

1	Robot moves	The robot's motion is estimated with odometry and displayed in Figure 2.
2	Get observations	The sensor gets a set of measurements and the results are displayed in Figure 3.
3, 4, 5	Compatibility	The robot checks which measurements match up to mapped features (Figure 6), trying three different methods (individual compatibility, nearest neighbour, JCBB (Joint Compatibility Branch and Bound).
6	Matching	Using one of these methods (as selected in the program) a match set is decided on and these are displayed in Figure 7.
7	Update	The robot and feature positions are updated, and any new features are added to the map (Figure 2).

After all of these actions, one full step is complete and the program cycles back to action 1.

6 Things to Try

The file `slam.m` contains various parameter settings which can be adjusted to produce different behaviour. You can edit this file either within Matlab or using an external text editor if you prefer.

6.1 Continuous Operation

`configuration.step_by_step`: when set to 1 the program will pause after every movement step; set it to zero and the program will run continuously right to the end. You will see the robot come right round the square corridor and 'close the loop' at the end. Observe how the uncertainty in the positions of new features (and in the position of the robot) increases as the robot moves around the loop, but then shrinks back down once the robot manages to re-observe early features and 'close the loop'. This is classic SLAM behaviour.

6.2 Odometry Only

Just for this section, type `slamodom` to run a version of the `slam` function where I have commented out all sensor measurements and updates — all that happens is that the robot estimates it position using odometry. Look at how it drifts away from the square (as in the first practical we did in week 2).

6.3 Coping with Clutter

One of the difficult issues in SLAM is feature matching (this is also called data association): when the sensor measures the position of a point feature, how can the robot be sure which of the features in its map it is observing? Or another possibility might be that the sensor is responding to something which is not a mappable feature such as a person walking in front of the robot.

This simulator has a sophisticated way of checking which measurement corresponds to which feature, called JCBB (Joint Compatibility Branch and Bound). Essentially, it looks for a set of correspondences between the measurements and mapped features which are consistent with a single motion of the robot and a static set of features. Try changing the line:

```
H = JCBB (prediction, observations, compatibility);
```

in `slam.m` to:

```
H = NN (prediction, observations, compatibility);
```

This replaces JCBB with a much simpler 'Nearest Neighbour' matching test. The mapping performance should become much worse as matching errors occur.

`configuration.people`: set to 1 and rogue measurements will be reported by the sensor, corresponding to the effect of people walking in front of the robot. Change back to the JCBB test and it will still sift out the true static features and build a sensible map because the moving people do not fit the rigid world model.

6.4 Measurement Uncertainty

It is possible to change the parameters of the sensor to simulate a range-finder which is more or less accurate, or has a different range. Try increasing the measurement standard deviations and see how the feature ellipses change.

`sensor.srho`: the standard deviation of range measurements from the sensor.

`sensor.stita`: the standard deviation of angular measurements.

`sensor.range`: the depth range of the sensor.

`sensor.minangle`, `sensor.maxangle`: the angular range of the sensor.