69156 Simultaneous Localization and Mapping

Lab 2: Data association for Karel in 2D

1 Problem

Imagine that the robot Karel is upgraded to live in a 2D space populated with static beepers placed at unknown locations. Karel is now equipped with an odometer that provides measurements of its own 2D motion. Karel also as a sensor that detects beepers that respond to Karel honking from within certain angle and distance ranges. The sensor can estimate the range and bearing to said beepers. There are also other robots that move around in Karels world. These also respond to Karels sensor within a certain range. Karel cannot distinguish whether its a beeper or another robot responding.

2 Starting the Simulation

Download the skeleton code from Moodle2, unpack the file and run Matlab. In the main Matlab window, type `karel2d_lab2` to start the simulation. This starts the main function defined in `karel2d_lab2.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 `karel2d_lab2` to start again).

3 Displays

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

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

4 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 different methods is the goal of this partical.
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.

5 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.

`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.

`configuration.odometry`: set to 0 and odometry will not be available, so the data association algorithm will have to work with large motion error and uncertainty.

6 Tasks

1. Try **NN**. Analize the results.
2. Complete **SINGLES** and try it. Include people and try **SINGLES**. Analize the results.
3. Try **NN** and **SINGLES** without odometry. Analise the results.
4. Program **JCBB** and try it. Analise the results.
5. Have some time in your hands? Implement map maintenance: eliminate all features seen only once (why this?), more than two steps ago. Is this always a good idea?
6. Have some more time in your hands? Randomize Joint Compatibility, combining ideas from **RANSAC** and **JCBB** (exercise 5 or 6). *Tip:* Try to use the idea, not necessarily implementing the exact same algorithms described in the papers. **
7. Still some time in your hands? Improve the computational cost of NN and SINGLES. *Tip:* reducing the big O() is the idea worth more of your time (exercise 3). *** This is conceptually rather simple, but the implementation in matlab can be tricky.

7 Results

Do as much as you can and have time for. I expect that you should be able to complete up to task 4 (for a maximum grade of 8). Additional stuff is welcome (for example exercises). Write and submit a full, well-detailed (but with no unnecessary detail) report plus code by **February 24**. *: Some difficulty. **: More difficult. ***: Well, really difficult.