ENMT482 Assignment 1

The lab and the assignment will be done in pairs. It would be to your advantage to ensure that at least one of you is comfortable with Python or Matlab programming. Note, the particle filter part of the assignment is likely to take less time and has fewer marks than the sensor fusion part.

Part A (Sensor fusion)

I have replicated the 1-D robot example in Lecture 1 using a Turtlebot robot. It has an assortment of range sensors pointing at a wall located at $x = 0$ m, and can be commanded to move back and forth along the x-axis up to about $x = 3$ m.

There are six sensors:

- sonar1 is an HC-SR04 sonar rangefinder (0.02–4 m)
- sonar2 is a MaxBotix HRUSB-MaxSonar-EZ MB1433 sonar rangefinder (0.3–5 m)
- ir1 is a Sharp GP2Y0A02YK0F infrared rangefinder (0.15–1.5 m)
- ir2 is a Sharp GP2Y0A41SK0F infrared rangefinder (4–30 cm)
- ir3 is a Sharp GP2Y0A21YK infrared rangefinder (10–80 cm)
- ir4 is a Sharp GP2Y0A710K0F infrared rangefinder (1–5 m)

Note that aside from sonar2, all of these sensors are knock-offs and thus their actual performance characteristics may not match their specifications.

Your task is to choose three of the sensors (not including sonar2) and use the measurements from these three sensors along with the commanded speed to estimate the robot's position. You are provided with three datasets as CSV files, each containing true position, commanded velocity, and the sensor measurements. From the dataset 'calibration.csv' you can determine the probabilistic sensor models; from the datasets 'training1.csv' and 'training2.csv' you can determine the probabilistic motion models.

Additionally, you are provided with a test dataset which does not contain the true position, which you will use for your report. Some Python example code is provided, but you may use another language if you prefer (although we may not be able to help).

To fuse the data well you will need to determine a probabilistic sensor model for each sensor and a probabilistic motion model for the Turtlebot. You can use whatever Bayes filter you like for your sensor fusion (except a particle filter) but you must justify your choice.

Hints:

1. Each sensor has an observation model of the form:

   $$Z = h(X) + V(X),$$

   so you should determine $h(X)$ and $f_V(v; x)$ for each sensor from the calibration data.

2. First plot the measured data as a function of x.

3. Postulate a function, $h(x)$, to fit the data. A good fit should have a zero-mean error.

4. Some sensors have many outliers that you will need to exclude. You may need to use iteration.
5. You can determine the variance of V from the residuals between the measured and modelled data.
6. To determine the motion model, I suggest estimating the measured speed and comparing this with the commanded speed.

Part B (Particle filter)

I have replicated the example in Lecture 12 using a Turtlebot robot, with fiducial markers for the beacons. You can see some of these markers placed around the Automation lab and the hallway, along the route we used for the lab. An image of one of these markers can be used to estimate the six-degree-of-freedom transformation between the robot and the marker, but for our purposes I've just kept x, y and rotation. The variance of the estimate is approximately proportional to the square of the distance between the robot and the marker.

Your task is to implement a particle filter which uses the beacon observations and either odometry and/or commanded velocity/rotation rate to estimate the robot's location. You are provided with one dataset as a CSV file, containing a "true" position from SLAM, estimated positions from odometry, the commanded forward speed and rotation rate, and the ID and position of any beacon observations. When two beacons are visible to the camera, there are two rows with the same time and position but different beacon observations. Additionally, you are provided with a CSV file containing the location of each beacon. Again, some Python example code is provided (I had to restrain myself to not use classes).

If you wish, you may use the true position to initialise your particles, but you must not use it after that. For extra credit, make your implementation robust to unknown starting positions.

In the video on Learn, I'm using the range-bearing sensor model from the lecture notes, and my motion model applies the difference between consecutive odometry positions to each particle with some process noise.

The SLAM algorithm used is RTABMAP, which is generally a bit better than `gmapping` but gets confused around the identical desks. In the example code I've just dropped the bits where it jumps around; it's pretty good the rest of the time.

The SLAM algorithm estimates a 2-D pose for the Turtlebot $(x, y; \theta)$, where x and y are in metres and θ is in radians. The pose is of the Turtlebot's base-frame with respect to the global map-frame. The heading angle, θ, is measured anti-clockwise from the x-axis of the map-frame.

The odometry is estimated from a wheel encoder for each wheel and a gyro using an EKF. Like the SLAM estimates, the odometry provides 2-D poses of the Turtlebot's base-frame in the map-frame.

The poses of the beacons are estimated in the camera-frame of the Turtlebot. This camera-frame is offset from the base-frame of the Turtlebot. Full 6-D poses are measured but only $(x, y; \theta)$ are given. Again x and y are in metres and θ is in radians. The camera-frame x

direction is in the Turtlebot's forward direction of travel, the y direction is to its left, and θ is measured anti-clockwise from the x direction of the base-frame.

For your particle filter to work:

1. You will need to understand the difference between the global (map) coordinates and the local (robot) coordinates and how to transform between the two. It is easy to get your coordinate transformations confused so I suggest plotting some data points on graph paper to check.
2. An accurate motion model. I recommend testing this first without the sensor model. The particles should move in the correct direction and spread out with time.
3. An accurate sensor model. Unfortunately, there is no calibration data so some trial and error is required. A range standard deviation of 0.1 m and an angle standard deviation of 0.1 radian will get you in the ballpark.

A pose $(x_1, y_1; \theta_1)$ in coordinate frame 1 can be converted to a pose $(x_2, y_2; \theta_2)$ in coordinate frame 2 using

$$\begin{aligned} x_2 &= x_1 \cos\Delta\theta - y_1 \sin\Delta\theta + \Delta x, \\ y_2 &= y_1 \cos\Delta\theta + x_1 \sin\Delta\theta + \Delta y, \\ \theta_2 &= \theta_1 + \Delta\theta, \end{aligned}$$

where Δx and Δy are the translations of frame 1 with respect to frame 2 and $\Delta\theta$ is the anti-clockwise rotation of frame 1 with respect to frame 2.

The inverse transformation is

$$\begin{aligned} x_1 &= (x_2 - \Delta x)\cos\Delta\theta + (y_2 - \Delta y)\sin\Delta\theta, \\ y_1 &= (y_2 - \Delta y)\cos\Delta\theta - (x_2 - \Delta x)\sin\Delta\theta, \\ \theta_1 &= \theta_2 - \Delta\theta. \end{aligned}$$

Note, when calculating differences between angles it is necessary to choose the smallest angle in the range $-\pi \leq \theta < \pi$.

Part C (Turtlebot Lab)

You need to use the Turtlebots in the Automation Lab during a booked session to generate a map of the Lab and corridor.

Report and Code Submission

By the submission deadline you need to submit your code, along with one short report (five A4 sides max, 12pt, excluding title page and optional one-page appendix) and a peer assessment each (half A4 side max).

Most of the marks are for the report; a small number of bonus marks are available for good code and results, but not enough to make up for a rushed report. A well-written report detailing exactly how you performed each stage of the assignment (using equations, diagrams, and graphs where appropriate) but got poor results will get a better mark than a thrown-together report with perfect results and beautiful code.

See the report template for what the report should include.

The peer assessments should include:

- Your group number.
- A summary of what you (personally) did.
- How you would divide 20 bonus marks between yourself and your partner.

All submissions are through Learn. Please submit the report and peer assessment as PDF files. Please submit a zip file of your code alongside your report.

Assessment (provisional)

Each of the following is marked equally:

1. Part A sensor models
2. Part A motion model
3. Part A Bayes filter
4. Part A results
5. Part A discussion
6. Part B sensor models
7. Part B motion models
8. Part B implementation
9. Part B results
10. Part B discussion
11. Part C map and gmapping observations

In addition, marks are allocated for:

1. Report presentation
2. Report writing