ENMT482 Assignment 1

The lab and the assignment will be done in pairs. Note, Part A is likely to take more time and has more marks than Part B and Part C.

Part A (Extended Kalman filter)

On the planet Hank, Khan wanted to set up a ice-cream delivery business using flying drones. The drones would vertically take-off, fly with a constant speed, at a constant height, in a straight line to the desired destination before descent. Hank has no global positioning system, so to localise the drone position, Khan decided to use two radio beacons with sensors made by the Ankh corporation that output values proportional to the distance from the drone to the associated beacon.

To calibrate the Ankh range sensor, Khan borrowed a super-duper lidar to provide the ground truth. The measured results are stored in a file `ankh.csv`.

While designed to fly at constant speed, the drones are subject to random accelerations in the forward direction only (there are no angular accelerations). Khan performed a trial of a drone flying a straight path and measured the sensor outputs and ground truth. The results are stored in a file `training.csv`.

Your goal is to use an EKF to estimate the position and speed of the drone and compare the result with the ground truth. Code is provided in `ekf.py` to implement the EKF equations[1] and example code is provided in `ekfdemo.py`. You will need to:

1. Determine a sensor model using the data in `ankh.csv`. From this determine the $\mathbf{C}$ matrix, the $h(x)$ function, and the covariance matrix, $\mathbf{R}$, describing the sensor noise.
2. Determine a motion model using the data in `training.csv`. From this determine the $\mathbf{A}$ matrix, the $g(x)$ function, and the covariance matrix, $\mathbf{Q}$, describing the sensor noise.
3. Determine the path flown by the drone using an EKF and the known starting point.
4. Determine the path flown by the drone[2] using only the sensor data and an inverse sensor model.
5. Determine the path flown by the drone using dead-reckoning and the known starting point (do not use the measurements). You can determine the starting point from the first line of the training data. You can determine the initial speed from the first two lines of the training data.
6. Plot the errors in x-positions and y-positions as functions of time for the three approaches. Label and add a legend to your plots. Set the ranges to clearly show the salient details.
7. Determine the mean state and covariance estimated by the EKF at the end of the data set.

[1]See the course reader for details.
[2]You will need to find the intersection of two circles.

8. Discuss how the Kalman gains vary with time.
9. Compare the path estimated by the EKF, with the sensor-only, and dead-reckoning approaches.

Khan found some cheaper sensors on Nelliexpress made by the Nakh corporation. Their accuracy decreases with range from the beacon. They also produce outliers as can be seen in the calibration file `nakh.csv`. For these sensors, determine the sensor model, and then use them with an EKF to determine the path of a drone from the measured data in dataset `test.csv`. Note, this dataset does not have the ground truth. Discuss your results.

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.

   1. First plot the measured data as a function of x.
   2. Postulate a function, $h(x)$, to fit the data.
   3. Plot the residuals $z - h(x)$. A good fit should have a zero-mean error. Note, some sensors have outliers that you will need to exclude. You can do this iteratively to improve the estimate.
   4. Determine the variance of V from the residuals (use the `var()` function with any outliers removed).
   5. Assuming the sensor noise is independent between sensors, determine the sensor noise covariance matrix $\mathbf{R}$.

2. For the motion model process noise:

   1. Consider the effect of a random acceleration and the error it would produce in the motion model equations.
   2. Write the process noise as $\mathbf{W} = \mathbf{F}A$, where A is a random variable representing the unknown acceleration.
   3. Determine the process noise covariance matrix using $\mathbf{Q} = \mathrm{Covar}(\mathbf{W}\mathbf{W}^{\mathrm{T}}) = \mathbf{F}\mathbf{F}^{\mathrm{T}}\,\mathrm{Covar}(A)$. Note, this will have correlations. It assumes the mean error is zero.

There are more hints in the guide document.

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 (ten A4 sides max., 12pt, excluding title page) and a peer assessment each (half A4 side max.).

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. These are not hypothetical.

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)

The following categories are assessed (note, Part A requires more work than Part B):

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

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.