ENMT482 Assignment 1 Guide

M.P. Hayes

1 Part A

This part of the assignment is about localising the position of a robot in 2-D using an extended Kalman filter (EKF). The goal is to create sensor models, a motion model, and to use an EKF to improve the estimate. The better the model, the better the estimation.

For more details see Lecture 11, Multivariate Kalman filters, in the course reader.

1.1 Sensor models

You want a model of the form:

$$Z = h(r) + V(r), \tag{1}$$

where r is the range to the beacon and $V(r)$ describes the sensor noise.

1. Plot the sensor data using the file `ankh.csv` (see Python script `plot-ankh.py`).

2. Fit a parametric model to the data using parametric identification given measured pairs of data (r_n, z_n). The goal is to find the parameters that minimise the residuals:

$$v_n = z_n - h(r_n). \tag{2}$$

   You can fit your model by eye using trial-and-error to find the best parameters that produces residuals with zero mean and the minimum variance. Alternatively, you can use an optimiser. For example, here's some example Python code for a range sensor with a quadratic model:

```
from scipy.optimize import curve_fit

def model(r, a, b, c):

    return a + b * r + c * r * r


# Load data r, z

params, cov = curve_fit(model, r, z)

zfit = model(r, *params)

residuals = z - zfit
```

Note, that the `curve_fit()` function passes an array for the argument `r` so you have to return an array of the same size. If you have a piecewise model, you can do something like:

```
from scipy.optimize import curve_fit

def model(r, a, b, c):

    return a * (r < 1) + (b * r**2) * (r >= 1)
```

3. The tricky aspect of model fitting is dealing with outliers. One approach is to iteratively fit a model and then remove the obvious outliers which have residuals many times the standard deviation.

   In Python, you can use NumPy fancy indexing to create a new array that ignores the outliers, for example:

   ```
   residuals = data - model
   mask = abs(residuals) < std(residuals) * 5
   pruned_data = data[mask]
   ```

4. The residuals of a good model have zero mean with minimal variance. I suggest plotting the residuals, v_n, as a function of r_n to see how good your model is.

5. Determine the variance of the residuals to estimate σ_V^2. Assuming the noise is additive, this is the same as σ_Z^2.

For an EKF you need to adapt your sensor model to be a function of the state vector:

$$Z = h(\mathbf{X}) + V(\mathbf{X}). \tag{3}$$

Since there are two sensor measurements, they need to be combined into a vector. This results in a model of the form:

$$\mathbf{Z} = \mathbf{h}(\mathbf{X}) + \mathbf{V}(\mathbf{X}). \tag{4}$$

Finally, this needs to be linearised (see the lecture on multivariate Kalman filters in the course reader) and the covariance matrix determined. Note, you can assume the sensors are independent.

1.2 Motion model

Here you need a stochastic model to predict X_n from the previous state. For example,

$$\mathbf{X}_n = g(\mathbf{X}_{n-1}, u_{n-1}) + \mathbf{W}_n. \tag{5}$$

where $\mathbf{W}_n$ is the process noise random vector (to represent uncertainty in the model). The commanded speed u is unknown so your model needs to ignore it.

Assume a constant speed model, in which case any acceleration will produce an error. You need to determine $\mathbf{W}_n$ by determining what error in the model will be produced by a random acceleration. For the EKF, you need the covariance of the process noise. This is given by

$$\mathbf{Q} = \text{Covar}(\mathbf{W}\mathbf{W}^{\mathrm{T}}). \tag{6}$$

This equation assumes that the mean error is zero.

You can determine the variance of the acceleration from the training data, by twice differentiating position with respect to time and using the `var` function.

2 Part B

This part of the assignment is about localising the position of a robot in 2-D using fiducial markers (beacons) and a particle filter. These markers are sensed by a camera on a Turtlebot2 robot to estimate the local pose of the marker with respect to the robot.

The particle filter algorithm is written for you but you need to write motion and sensor models. If you are feeling clever, you might try adapting the number of particles and/or not using knowledge of the starting position.

More details can be found in the course reader. See Lectures 12 to 14.

2.1 Motion model

You can use either the velocity or odometry motion models. The latter has the advantage of decoupling the errors, see lecture notes.

To test your motion model, disable the sensor model (so that the particle weights do not change) and see if the particles move by the correct amount in the correct direction. A useful Python script is 'test-motion-model.py'.

Once the particles are moving correctly, add some random behaviour to the motion of each particle to mimic process noise. The amount of randomness depends on how good your motion model is. However, it is difficult to evaluate the process noise and I suggest that you tweak this by trial and error, starting with a small amount of noise.

2.2 Sensor model

To correctly implement the sensor model you will need to understand:

1. The difference between the robot and global (map) reference frames.
2. How to calculate the range and bearing of the beacons with respect to the robot given the estimated pose of the beacons.
3. How to calculate the range and bearing of the beacons with respect to each particle.
4. How to use the `arctan2` function.
5. How to determine the smallest angle between two vectors.

All these aspects are covered in the lecture notes.

Unfortunately, there is no calibration data for the fiducial marker sensor. I suggest modelling the sensor noise (in both range and bearing) as Gaussian random noise and choosing the standard deviation by trial and error. A range standard deviation of 0.1 m and an angle standard deviation of 0.1 radian will get you in the ballpark. Note, the standard deviation will vary with the observed pose of the marker (it will be more accurate front-on than side-on) and the distance to the marker.

2.3 Particle filter

The more particles you have, the better the estimate but the slower the computation.

Here's what I suggest you do:

1. Test motion model without adding noise to particles and disable sensor model. The particles should move in the correct direction.
2. Test motion model with added noise and disable sensor model. The particles should move in the correct direction but spread out.
3. Enable sensor model but with large standard deviations for the range and bearing errors. The particles should get closer together whenever a beacon is visible.
4. Reduce standard deviations in sensor model to get better tracking.

3 Miscellaneous

3.1 Calculating angle

In mathematics we determine the angle of a right-angled triangle from the opposite, y, and adjacent, x, distances using

$$\theta = \tan^1 \frac{y}{x}. \tag{7}$$

Numerically you need to use the arctan2 function (Python) or atan2 function (C), `theta = arctan2(y, x)`. This determines the correct quadrant and avoids singularities when x is zero.

3.2 Calculating angle between two angles

When calculating angle between two vectors we usually want the smallest difference. For example, let $\theta_1 = 315$ degrees and $\theta_2 = 45$ degrees. $\theta_1 - \theta_2 = 270$ degrees. However, this does not take the angle wrapping into account since the smallest angle between the vectors is 90 degrees.