Exercise 3 and 4 – Matching observations to maps, Perception – Sensor fusion (Kalman filter)

Ola Bengtsson

School of Information Science, Computer and Electrical Engineering, Halmstad University, P O Box 823, Halmstad, Sweden

1 Introduction

For a robot to become fully autonomous it must be able to localize itself, which is something that can be achived by dead reckoning, but as we saw in Exercise 2 the errors in the dead reckoning position grow without bounds, which means they are reliable over short distances only. Therefore the position of the robot must be corrected (calibrated) relative the environment in which the robot is currently active.

The exercise consists of laser data collected by the Snowhite robot, i.e. the three-wheeled robot used in exercise 2.

1.1 Plotting the Snowhite (and the laser data at a given location)

To plot a schematic image (the two rear wheels, its co-ordinate axis, the front wheel, the heading axis of the front wheel, the position of the laser sensor and (optional) the actual sensor readings at the given position) you can use the function `plot_threewheeled_laser(…)` which is defined and works as follows:

```
plot_threewheeled_laser(X, wdr, wbr, wtr, SteerA, wdf, wtf, L, alfa, beta, gamma, ANGLES, MEAS, view_meas)
```

For an explanation to the parameters; `X`, `wdr`, `wbr`, `wtr`, `SteerA`, `wdf`, `wtf` and `L` see exercise 2. The new parameters are; `alfa` = distance along the robots x-axis between the robots centre position and the centre position of the sensor (660mm), `beta` = distance along the robots y-axis between the robots centre position and the centre position of the sensor (0mm), `gamma` = the angular difference between the x-axis of the sensor and the x-axis of the robot ($-\pi/2$ radians). ANGLES and MEAS are [Nx1] vectors containing the laser data (angles [radians] and distances [mm]) and `view_meas` is a boolean value saying whether or not to plot the laser data.

2 Matching laser data to a set of line segments (the Cox scan matching algorithm)

Use the matlab script '`main.m`', which loads all laser data (angles and measured distances) together with the control input values (velocities and steering angles) and the reference model REF_MODEL consisting of line segment. The script predicts the position based on the control inputs (which are shown as blue dots in the figure), which are plotted together with the true positions (which are shown as black dots) and where the laser data are perceived (red circles).

1) Run the script and make sure you know what it does. Also make sure you understand how the laser works (e.g. by plotting some of the laser scans).

2) Complete the script so it also predicts the uncertainty of the dead reckoning position (which you did in exercise 2).

3) Write the function `Cox_LineFit(..)` (the Cox scan matching algorithm is described in the "Blanche - …" paper by Cox) in a separate m-file so it can be called in the following way:

```
[dx dy da C] = Cox_LineFit(angs, meas, [X(kk-1) Y(kk-1) A(kk-1)]', [alfa
beta gamma]', LINEMODEL);
```

4) Call your function from the main script and update your position estimates using the match results (dx, dy, da and C). Along the path, plot the errors and the estimated error variances.

5) (**This is exercise 4**) To see how the use of a Kalman filter affects the robot's position estimates, you should simulate the position fixes, i.e. for the moment not use the position fixes given by the scan matching algorithm (i.e. given by the function `Cox_LineFit(…)`). To simulate position fixes, simply add a zero mean Gaussian distributed noise (generated in Matlab by the function `randn`) to the *true positions* in those positions where the matching take place, which is done according to Equations 1 – 2.

$$\begin{aligned} \hat{x}_{pf} &= x_{true}(i) + \varepsilon_x \\ \hat{y}_{pf} &= y_{true}(i) + \varepsilon_y \\ \hat{\theta}_{pf} &= \theta_{true}(i) + \varepsilon_\theta \end{aligned} \tag{1}$$

In Equation 1, the errors ε_x, ε_y, ε_θ are uncorrelated and have zero mean and Gaussian distributions with standard deviations σ_x, σ_y, σ_θ, which gives the co-variance matrix (representing the uncertainty in the position fix) according to Equation 2.

$$\sum_{pf}(i) = \begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix} \tag{2}$$

The robot's position estimate, $\hat{X}(i)$, and co-variance matrix, $\sum_{\hat{X}}(i)$, at time i, are updated by the proper weighting of the position fix, i.e. by means of a Kalman filter, which is seen in Equation 3 and 4.

$$\hat{X}^+(i) = \sum_{pf}(i)\left(\sum_{pf}(i) + \sum_{\hat{X}}(i)\right)^{-1} \hat{X}^-(i) + \sum_{\hat{X}}(i)\left(\sum_{pf}(i) + \sum_{\hat{X}}(i)\right)^{-1} \hat{X}_{pf}(i) \tag{3}$$

$$\sum_{\hat{x}}^+(i) = \left(\sum_{pf}^{-1}(i) + \sum_{\hat{x}}^{-1}(i)\right)^{-1} \tag{4}$$

Run the entire sequence with encoder twice with different values of the error noise (i.e. vary the standard deviations (variances)), e.g. in the first run, use $\sigma_x = \sigma_y = 10$mm and $\sigma_\theta = 1°$. In the second run, instead use $\sigma_x = \sigma_y = 100$mm and $\sigma_\theta = 3°$, do you see any difference in the estimated positions?

6) (**This is exercise 4**) Instead of simply either using the dead reckoning position or the position from the matching algorithm fuse (using a Kalman filter) your dead reckoning position with your positions from the scan matching algorithm. The same as in 2.5, but using the position fixes given by the scan matching algorithm instead of simulating them.

3 Others

The reports (which are individual, although you are more than welcome to solve the exercises in smaller groups, e.g. by groups of two students) should contain all necessary equations with all assumptions motivated. You should also interpretive your results, i.e. handing in a plot without any explanations is not enough.