ME 580 Lab 3: Kalman Filter

Ahmed Elsaharti
Mechanical Engineering Department
University of North Dakota
ND, United States of America
ahmed.elsaharti@und.edu

***Abstract*—This report demonstrates the recreation of the Kalman filter for the application of a mobile robot navigating within an environment. This was done using MATLAB and results showing the filter succeeding were illustrated.**

I. Robot's state vector

A typical state vector for a mobile robot like the one used for this experiment (Fig. 1) would be:

$$X_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$$

where x_t is the x position at time t, y_t the y position at time t and θ_t the angle between the x-axis and the robot's front at time t.

This state vector is chosen because, since this is a two-wheeled mobile robot, there shouldn't be any change in the z-axis position. The main important states to keep track of are the x and y positions and the angle/heading of the robot.

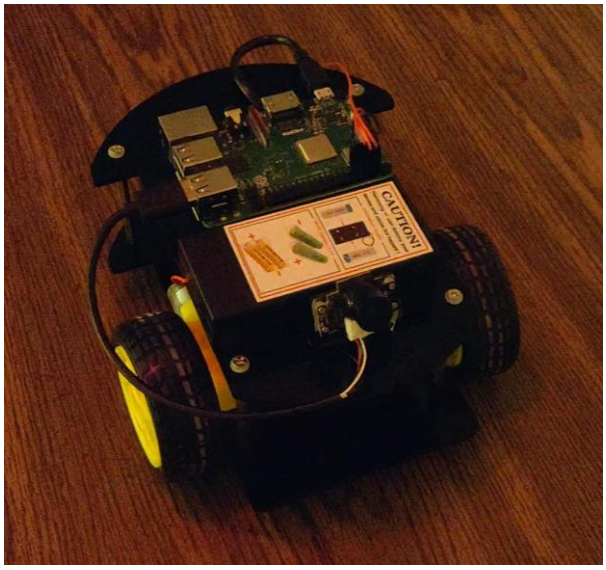

Fig. 1. Robot used in experiment

II. The Kalman filter

In this lab report, we implement a variation of the Kalman filter [1] used for mobile robots similar to the one we used.

There are multiple steps to this application of the Kalman filter (taken from [1]) and they are laid out and explained as follows:

1. Position prediction:

In this step we use trigonometrical math to generate a position prediction (a posterior) using the previous state vector of the robot (a prior). The previous state vector can be either an initial position prediction or a position prediction generated by the previous iteration that was Kalman filtered.

To do this we use

$$\hat{X}_t = \begin{bmatrix} \hat{x}_t \\ \hat{y}_t \\ \hat{\theta}_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \frac{\Delta S_r + \Delta S_l}{2} \cos\left(\theta_{t-1} + \frac{\Delta S_r - \Delta S_l}{2b}\right) \\ \frac{\Delta S_r + \Delta S_l}{2} \sin\left(\theta_{t-1} + \frac{\Delta S_r - \Delta S_l}{2b}\right) \\ \frac{\Delta S_r - \Delta S_l}{b} \end{bmatrix}$$

where $\begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix}$ is the state vector of our previous state (the prior), ΔS_r is the displacement of the right wheel (between the current and the previous states), ΔS_l is the displacement of the left wheel and b is the distance between both wheel along the wheel axis.

2. Covariance matrix update:

In this step we update the covariance that represents the uncertainties. This is done using

$$\hat{P}_t = F_x P_{t-1} {F_x}^T + F_u Q_t {F_u}^T$$

where P_{t-1} is the covariance of the previous state, F_x and F_u are jacobians and Q_t is the noise covariance associated with the motion model.

For this experiment we initially set P_{t-1} to $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and then as we progress the state covariance gets calculated to be put into the next iteration as a prior covariance.

F_x, F_u and Q_t can be calculated as follows:

$$F_x = \begin{bmatrix} 1 & 0 & -\Delta S_t sin\left(\theta_t + \frac{\Delta\theta_t}{2}\right) \\ 0 & 1 & \Delta S_t cos\left(\theta_t + \frac{\Delta\theta_t}{2}\right) \\ 0 & 0 & 1 \end{bmatrix}$$

$$F_u = \begin{bmatrix} \frac{1}{2}\cos\left(\theta_t + \frac{\Delta\theta_t}{2}\right) - \frac{\Delta s_t}{2b}\sin(\theta_t + \frac{\Delta\theta_t}{2}) & \frac{1}{2}\cos\left(\theta_t + \frac{\Delta\theta_t}{2}\right) + \frac{\Delta s_t}{2b}\sin(\theta_t + \frac{\Delta\theta_t}{2}) \\ \frac{1}{2}\sin\left(\theta_t + \frac{\Delta\theta_t}{2}\right) + \frac{\Delta s_t}{2b}\cos\left(\theta_t + \frac{\Delta\theta_t}{2}\right) & \frac{1}{2}\sin\left(\theta_t + \frac{\Delta\theta_t}{2}\right) - \frac{\Delta s_t}{2b}\cos\left(\theta_t + \frac{\Delta\theta_t}{2}\right) \\ \frac{1}{b} & -\frac{1}{b} \end{bmatrix}$$

$$Q_t = \begin{bmatrix} k_r|\Delta s_r| & 0 \\ 0 & k_l|\Delta s_l| \end{bmatrix}$$

where θ_t is the posterior angle or the third element of $\hat{X}_t$,

$$\Delta\theta_t = \frac{\Delta S_r - \Delta S_l}{b},$$

$$\Delta S_t = \frac{\Delta S_r + \Delta S_l}{2}$$

and k_r, k_l are error constants that are set to 0.1 in our case.

3. Observations:

These are the lidar readings/points our robot takes of the environment surrounding it. These get lines fit to them (using a line segmentation and fitting algorithm). Since these lines are taken by a sensor onboard the robot, they're generated in the robot frame $\{R\}$.

These are stored along with their covariances as

$$z_t^i = \begin{bmatrix} \alpha_t^i \\ r_t^i \end{bmatrix}, R_t^i = covariance\ matrix$$

Where z_t^i are the parameters of the 'i'th observed line during time 't' and R_t^i is the covariance matrix of the line.

4. Measurement predictions:

In order to be able to compare lines we expect to see based on our current position and those observed, we need to convert the parameters of the lines to the same coordinate frame. This is done because as opposed to the sensor data, the map data is in the world frame $\{W\}$.

We will be transforming lines z_t^j from the stored map in $\{W\}$ to $\{R\}$ as follows:

$$\hat{z}_t^j = \begin{bmatrix} {}^{\{w\}}\alpha_t^j - \hat{\theta}_t \\ {}^{\{w\}}r_t^j - \left(\hat{x}_t\cos\left({}^{\{w\}}\alpha_t^j\right) + \hat{y}_t sin\left({}^{\{w\}}\alpha_t^j\right)\right) \end{bmatrix}$$

$$H^j = \begin{bmatrix} 0 & 0 & -1 \\ -\cos({}^{\{w\}}\alpha_t^j) & -\sin({}^{\{w\}}\alpha_t^j) & 0 \end{bmatrix}$$

5. Matching:

To find matches, each of the lidar lines is checked with the lines on the saved map. For an observation to be valid or matched this constrain has to be true:

$$V_t^{ij^T}\left(\varepsilon_t^{ij}\right)^{-1}V_t^{ij} \le g^2$$

Where

$$V_t^{ij} = \begin{bmatrix} \alpha_t^i \\ r_t^i \end{bmatrix} - \begin{bmatrix} {}^{\{w\}}\alpha_t^j - \hat{\theta}_t \\ {}^{\{w\}}r_t^j - \left(\hat{x}_t\cos\left({}^{\{w\}}\alpha_t^j\right) + \hat{y}_t sin\left({}^{\{w\}}\alpha_t^j\right)\right) \end{bmatrix}$$

Which is the difference between the 'i'th observation and the 'j'th line in $\{R\}$ and

$$\varepsilon_t^{ij} = H^j\hat{P}_t H^{j^T} + R_t^i$$

6. Kalman filter estimate:

In this final step the matrices calculated are used to find an estimate of the location and pose of the robot using the Kalman filter.

For valid/matched observations, the values for V_t^{ij}, z_t^i, H^j and R_t^i are stacked as follows:

$$Z_t = \begin{bmatrix} z_t^i \\ z_t^{i+\cdots} \\ \vdots \end{bmatrix}, V_t = \begin{bmatrix} V_t^{ij} \\ \vdots \\ \vdots \end{bmatrix}, H_t = \begin{bmatrix} H^j \\ H^{j+\cdots} \\ \vdots \end{bmatrix}$$

and $R_t = blkdiag(R_t^i, R_t^{i+\cdots}, \ldots)$.

These values are used in the following equations to be able to finally obtain an estimate of the current state vector parameters:

$$\varepsilon_{IN_t} = H_t \cdot \hat{P}_t \cdot H_t^T + R_t$$

$$K_t = Kalman\ gain = \hat{P}_t H_t^T\left(\varepsilon_{IN_t}\right)^{-1}$$

$$X_t = Kalman\ filter\ prediction = \hat{X}_t + K_t V_t$$

$$P_t\ (for\ the\ next\ iteration) = \hat{P}_t - K_t\varepsilon_{IN_t}K_t^T$$

III. Used Parameters

The parameters used in the MATLAB script used to create the Kalman filter used in this report include:

a. Error constants Kr and Kl = 0.1
b. Initial covariance matrix of zero
c. Split and merge algorithm maximum distance of 10mm
d. Split and merge algorithm merging angle threshold of 0.2radians

e. Split and merge algorithm merging radius threshold of 100mm

These were determined using trial and error to get the best results. The next section will be showing how a change in the initial covariance matrix affects results in our case.

IV. Results

In this section we will be looking at multiple results showing how the application of the developed Kalman filter helps in position estimation within the environment shown in Fig. 2.

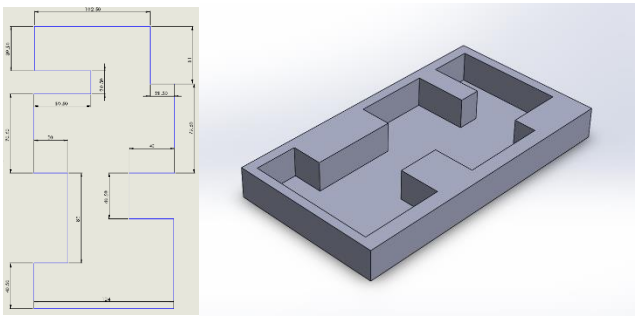

Fig. 2. 2D and 3D models of the environment

The files attached with this submission are readings taken as the robots moves along a straight line 50mm at a time.

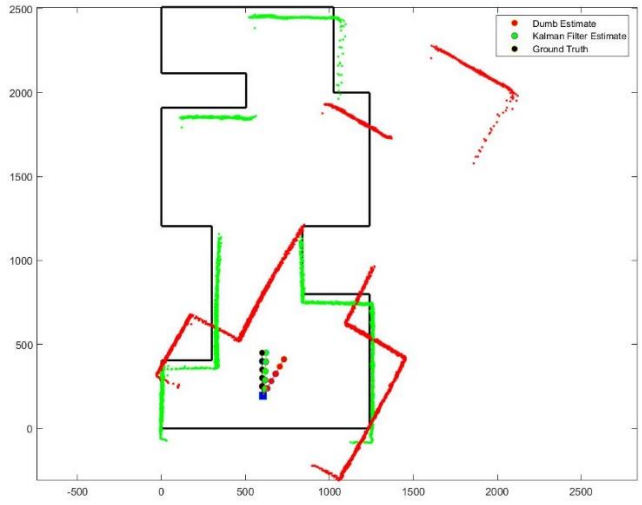

Fig. 3. Zero initial covariance and a good initial position with initial angle at 60

Fig. 3 shows a run with an initial covariance of zero and an accurate initial position with an initial angle of 60 rather than the actual 90. The filter manages to recover and provide an estimate very close to the ground truth. Fig 4. shows how our Kalman filter manages to recover from a bad prior with inaccurate initial x, y and theta values.

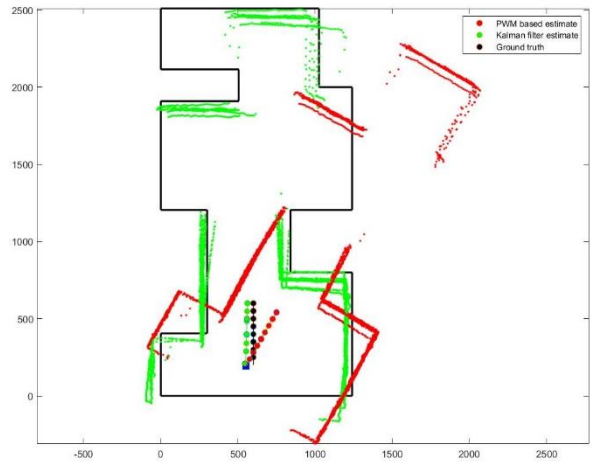

Fig. 4. Zero initial covariance and a bad initial position with initial angle at 60

Finally, Fig. 5 shows how the filter fails when the initial covariance matrix is set to 0.01 while Fig. 6 shows the filter working well with a covariance of 100.

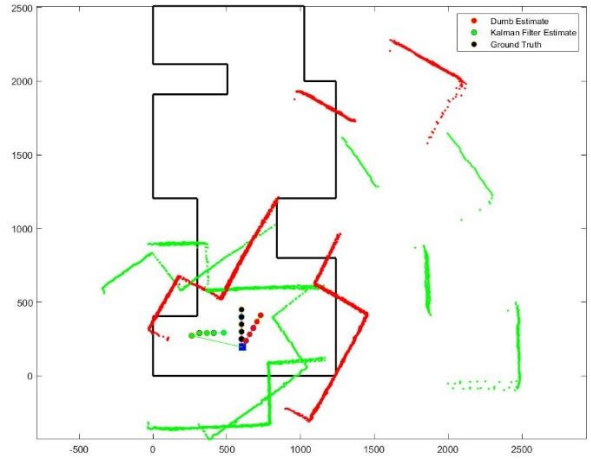

Fig. 5. Filter failing when covariance matrix is set to 0.01

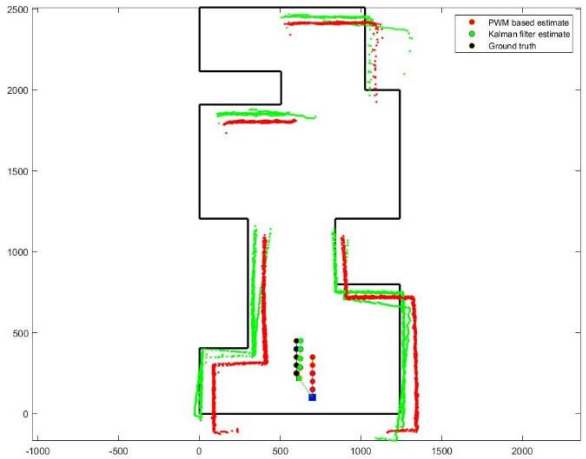

Fig. 6. Filter succeeding when covariance matrix is set to 100

REFERENCES

[1] Siegwart, Roland, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots, second edition.