ME8135 - State Estimation - Assignment 2

Austin Vuong

May 2023

1 Kalman Filter

Use pyGame, or any other similar libraries, to simulate a simplified 2D robot and perform state estimation using a Kalman Filter. Motion Model:

$$\dot{x} = \frac{r}{2}(u_r + u_l) + w_x \quad , \quad \dot{y} = \frac{r}{2}(u_r + u_l) + w_y \tag{1}$$

$r = 0.1$ m, is the radius of the wheel, u_r and u_l are control signals applied to the right and left wheels. $w_x = N(0, 0.1)$ and $w_y = N(0, 0.15)$

Simulate the system such that the robot is driven 1 m to the right. Assume the speed of each wheel is fixed and is 0.1 m/s Use these initial values

$$x_0 = 0 \ , \ y_0 = 0 \ , \ P_0 = \mathbf{\Sigma} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \text{ (initial covariance matrix)} \tag{2}$$

and assume the motion model is computed 8 times a second. Assume every second a measurement is given:

$$z = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} r_x \\ r_y \end{bmatrix} \tag{3}$$

where $r_x = N(0, 0.05)$ and $r_y = N(0, 0.075)$

Deliverable: Plot and animate the trajectory along with covariance ellipse for all motion models and measurement updates. Report what happens when a measurement arrives. Note: you need to discretize the system: $x = (x_k - x_{k-1})/T$, where T is 1/8 sec. In your animation, show both ground-truth and estimates.

1.1 Report

Note: I have changed to the farthest right matrix in equation 3 to be compatible with the left matrix. I also assume the second given r_x was intended to be r_y.

I have made note of design decisions in the appendix to this document.

When a measurement arrives it is incorporated into the prior predictions of the mean and covariance. To do this we find the innovation and the Kalman gain. The innovation is the difference between the actual measurement and the measurement expected by our predictions.

$$\text{innovation} = Cx - Cx_{pred}$$

where C is the measurement matrix, and x is the state, and x_{pred} is our predicted state based on our motion model.

The Kalman gain represents weights on how much to prefer the measurement or predictions based on the respective covariances. Intuitively, the Kalman gain gives greater weight to the estimates with less uncertainty for each state parameter. It is given as:

$$K = P_{pred}C^{\top}(CP_{pred}C^{\top} + Q)^{-1}$$

where P_{pred} is our covariance from predictions and Q is the measurement covariance.

From these we can compute an updated state by weighting the innovation with the Kalman gain.

$$x_{pred} \leftarrow x_{pred} + K \times (\text{innovation})$$

The updated covariance is also found as a weighting based on the Kalman gain.

$$P_{pred} \leftarrow (\mathbf{I} - KC)P_{pred}$$

In this setup we can see that the uncertainty of the prediction grows until a measurement arrives. At those points the measurement covariance is smaller than our prediction covariance and the Kalman gain gives precedence to the measurements accordingly.

This system reaches a steady state Kalman gain after a few iterations.

2 Extended Kalman Filter

Repeat the previous assignment, this time with a classic motion model and range observations made from a landmark located at $M = [10, 10]$. L is the distance between the wheel, known as wheelbase, and is 0.3m.

$$\dot{x} = \frac{r}{2}(u_r + u_l)\cos(\theta) + w_x \quad , \quad \dot{y} = \frac{r}{2}(u_r + u_l)\sin(\theta) + w_y \quad , \quad \dot{\theta} = \frac{r}{L}(u_r - u_l) \tag{4}$$

Assume

$$u_\omega = \frac{1}{2}(u_r + u_l) \quad , \quad u_\psi = (u_r - u_l) \tag{5}$$

Then the equations become:

$$\dot{x} = ru_\omega \cos(\theta) + w_\omega \quad , \quad \dot{y} = ru_\omega \sin(\theta) + w_\omega \quad , \quad dot\theta = \frac{r}{L}u_\psi + w_\psi \tag{6}$$

$w_\psi = N(0, 0.01)$ and $w_\omega = N(0, 0.1)$. Program the robot such that it loops around point M.

(a) Compute the EKF with the linear measurement model in the previous assignment.

(b) Compute the EKF with range/bearing measurements of point M. Assume range noise is N(0,0.1) and bearing noise is N(0,0.01). Range is in meters, and bearing is in radians. Visualize the measurements as well.

2.1 Report (a)

The general setup is similar to Part 1 and follows a "cookbook" construction of the Kalman filter. The robot's state is now a 3-vector with the angle, θ, appended. The other matrices are adjusted to be compatible. The new motion model is now non-linear and so the motion matrix is adapted with an update function to recompute the matrix using the angle. The measurement matrix, C, is extended from 2x2 to 2x3. This new column is filled with zeros as the measurement only reads position and not angle.

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$$

The impact of an incoming measurement is similar to Part 1. The key difference is that, as the measurement does not give an angle, the predicted angle is solely dependant on the motion predictions. As such the predicted covariance continues to grow as the simulation runs.

2.2 Report (b)

First, we establish the formulas for range, ρ, and bearing, ϕ from point M:

$$\rho = \sqrt{X^2 + Y^2}$$
$$\phi = \arctan \frac{Y}{X}$$

Where $X = M_x - \text{state}_x$ and $Y = M_y - \text{state}_y$.

To adapt the KF for non-linear measurements we replace the measurement matrix, C, with an appropriate Jacobian:

$$C_J = \begin{bmatrix} \frac{\partial\rho}{\partial X} & \frac{\partial\rho}{\partial Y} & 0 \\ \frac{\partial\phi}{\partial X} & \frac{\partial\phi}{\partial Y} & 0 \end{bmatrix}$$

where the matrix elements are as follows:

$$\frac{\partial \rho}{\partial X} = \frac{Y}{\sqrt{X^2 Y^2}}$$

$$\frac{\partial \rho}{\partial Y} = \frac{X}{\sqrt{X^2 Y^2}}$$

$$\frac{\partial \phi}{\partial X} = \frac{X}{X^2 Y^2}$$

$$\frac{\partial \phi}{\partial Y} = \frac{Y}{X^2 Y^2}$$

Now this new measurement matrix can be substituted into the Kalman gain equation giving us the EKF.

3 Appendix

3.1 Design Choices

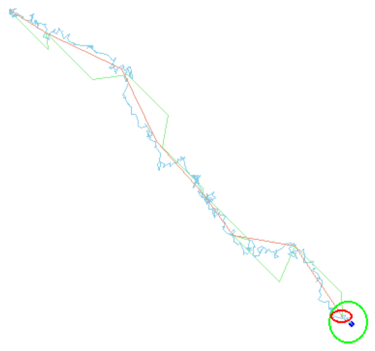
In terms of coordinates, the world a scale factor is applied to coordinates when drawing and an offset is also applied to allow for moving of the view. In Part 1 the world is scaled so that (0, 0) is the top left of the window and (1, 1) is the bottom right and the offset is $\mathbf{0}$. In Part 2 the offset centers point M in the middle of the window and the scale factor is adjusted for a more zoomed out view.

For the visualization: blues represent the ground truth, greens are the motion predictions, and reds show the measurements. Lighter colors are used for the trajectories. The world, ellipses, and trajectories are drawn to separate surfaces so they can be toggled as needed in the future.

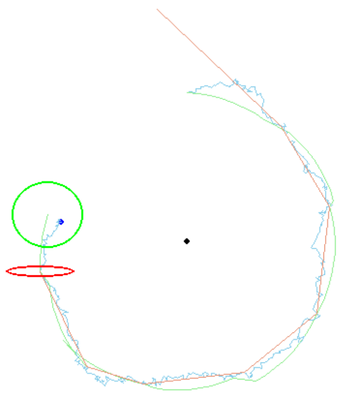
The timing of predictions and updates the measurements are manged by pyGame timers set to trigger at the specified intervals. Meanwhile the ground truth velocity is computed (with noise) every frame. Velocities are integrated into the state as the familiar, $\mathrm{x}_k = \mathrm{x}_{k-1} + \dot{\mathrm{x}}_k dt$, where dt is the interval between frames. Sampling noisy velocity at every frame results in the ground truth moving erratically and makes for a clear contrast between the visualized elements. For Part 2, the control inputs are scaled up so that the robot can make at least one orbit with the given noise.

3.2 Images

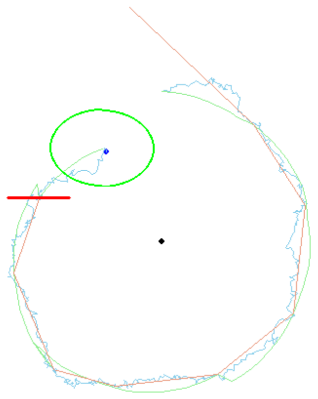
Below are screenshots of each part for inspection without running the code.



(a) Part 1.



(b) Part 2(a).



(c) Part 2(b).

Figure 1: Screenshots from each part.