2D Robot motion estimation using Kalman Filter

1st Ali Babaei
Robotics and Computer vision Lab
Ryerson University
Toronto, Canada
ali.babaei@ryerson.ca

***Abstract*—This document shortly reports simulation results of motion estimation of a robot in 2D using Kalman Filter with a simplified motion model.**

***Index Terms*—Kalman Filter, motion estimation, 2D , robot pose**

I. Introduction

The Kalman filter is a recursive estimator. This means that only the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current state. In contrast to batch estimation techniques, no history of observations and/or estimates is required. [1] The Kalman filter has two main stages: Prediction stage, and a correction stage. For the prediction state, we predict the state of the object as well as the covariance matrix. Here two functions implemented; predict() and update().

II. Problem Statement

The goal is to simulate the motion of a robot in 2D environment with Kalman Filter, using a simplified motion model. The simulation is implemented in python using pygame package.

We are given a simple control signal for left and right wheel,the radius of wheel is 0.1 m, and the speed is 0.1 m/s. As shown in Equation (1), there are also noise in the process with zero mean and 0.1 and 0.15 standard deviation for x and y, respectively.

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

The robot starts at [0,0] and the initial covariance and variances are also zero. The motion model is computed with 8Hz rate, and one measurement is given every second with the following model:

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

the measurement noise is normally distributed with zero mean and 0.05 and 0.075 standard deviation.

III. Solution

For our task, we will track a robot in a 2D space, such as a field. We will start with a simple noisy sensor that outputs noisy (x, y) coordinates which we will need to filter to generate a 2D track. That is, at each time T it will provide an (x, y) coordinate pair specifying the measurement of the sensor's position in the field.

Step 1: Choose the State Variables As always, the first step is to choose our state variables. We are tracking in two dimensions and have a sensor that gives us a reading in each of those two dimensions, so we know that we have the two observed variables x and y. x=[x , y]. as we are using a simplified motion model, we will discard the velocity and use only positional coordinates.

Step 2: Design State Transition Function Our next step is to design the state transition function. The state transition function is implemented as a matrix F that we multipy with the previous state of our system to get the next state, like so. x'=Fx

Step 3: Design the Motion Function We have control inputs to our robot. So we set the motion input u to: u=[ur, ul]

Step 4: Design the Measurement Function. The measurement function defines how we go from the state variables to the measurements using the equation z = Hx. We use the Kalman filter to go from measurements to state. But the update step needs to compute the residual between the current measurement and the measurement represented by the prediction step. Therefore H is multiplied by the state x to produce a measurement z.

Step 5: Design the Measurement Noise Matrix In this step we need to mathematically model the noise in our sensor. For now we will make the simple assumption that the x and y variables are independent Gaussian processes. That is, the noise in x is not in any way dependent on the noise in y, and the noise is normally distributed about the mean. We use N(0,0.05) for x and N(0,0.075) for y. They are independent, so there is no covariance, and our off diagonals will be 0.

Step 6: Design the Process Noise Matrix Finally. Similar to the last step, the process noise is given by normal distributions N(0 , 0.1) and N(0 , 0.15).

Step 7: Design Initial Conditions. For our simple problem we will set the initial position at (0,0) with a velocity of (0,0). Since that is a pure guess, we will set the covariance matrix P to zero.

IV. Results

There are some practical considerations while implementing the KF algorithm: according to the literature [2], a normal distribution is shown by N(mean , covariance). So the values given in the assignment for wx, wy, rx, and ry are covariances. Hence while using Numpy.random.normal() to draw random

samples from these distributions one should notice that the it takes standard deviation rather than covariance as inputs. [3]

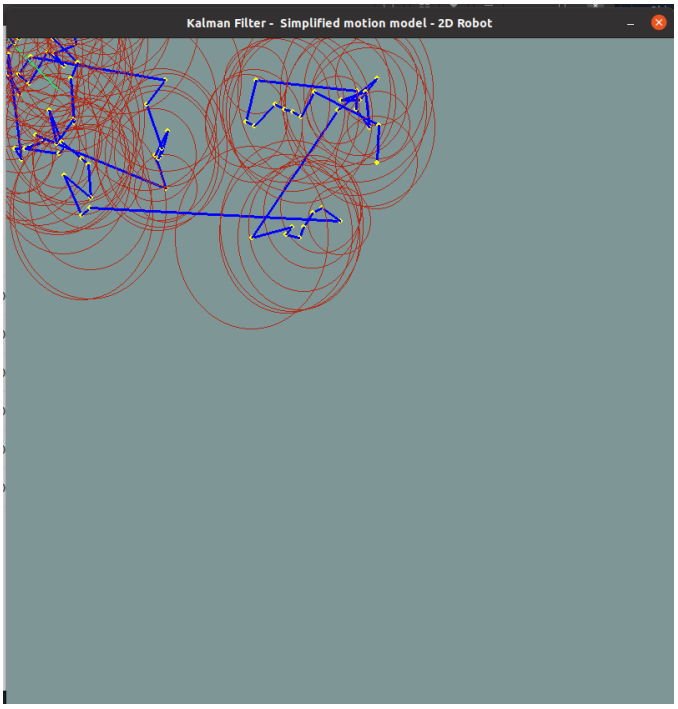

Fig. 1. Estimated 2D pose, speed=0.1 m/s

Considering these points, and given parameters the result of simulation is shown in the figure 1 . the green line is ground truth, the path of robot calculated with the control signal, given the condition that there is no noise. The yellow circle is robot position, the error ellipse is shown with red, and blue line connects the estimated position of robot at each time step.

the result shows the movement is quite random. It might be due to small SNR. To test this assumption we ran the code with a ur and ul 100 times larger. Figure 2 shows the result. The prediction is performed better but still a lot of variation is seen. One possible explanation is that the amount of noise is still large, compared to the signal. We ran the test another time, with the assumption that given wx, wy, rx, and ry are standard deviations, not variances. So they should be squared for R and Q matrices. As shown in Figure 3, the algorithm performed much better.

the implementation and more graphics of the results are available at *https://github.com/AliBabaei0/Estate-Estimation-for-Robotics-self-study*.

REFERENCES

[1] G. Bishop, G. Welch *et al.*, "An introduction to the kalman filter," *Proc of SIGGRAPH, Course*, vol. 8, no. 27599-23175, p. 41, 2001.

[2] T. D. Barfoot, *State Estimation for Robotics*, 1st ed. USA: Cambridge University Press, 2017.

[3] Numpy, "Description of random.normal() function in numpy," https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html, 2021, accessed: 2021-06-15.

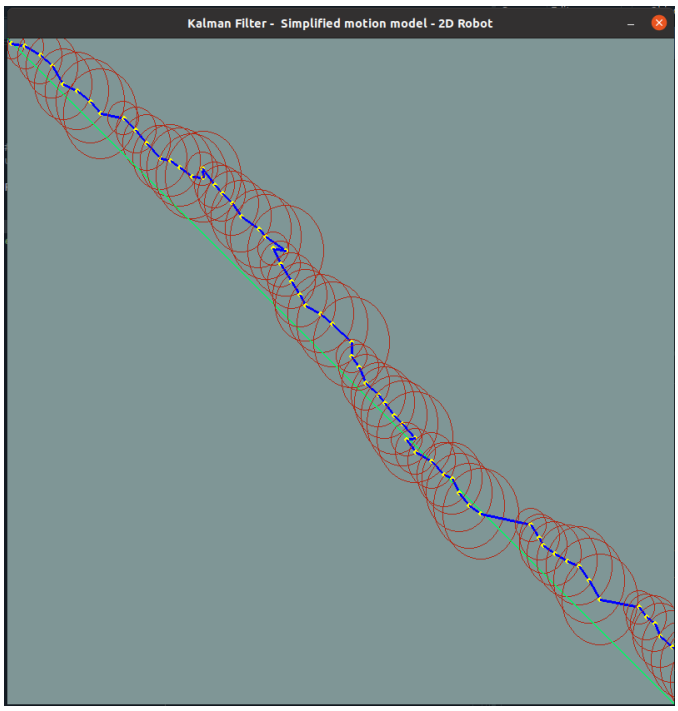

Fig. 2. Estimated 2D pose, speed=10 m/s

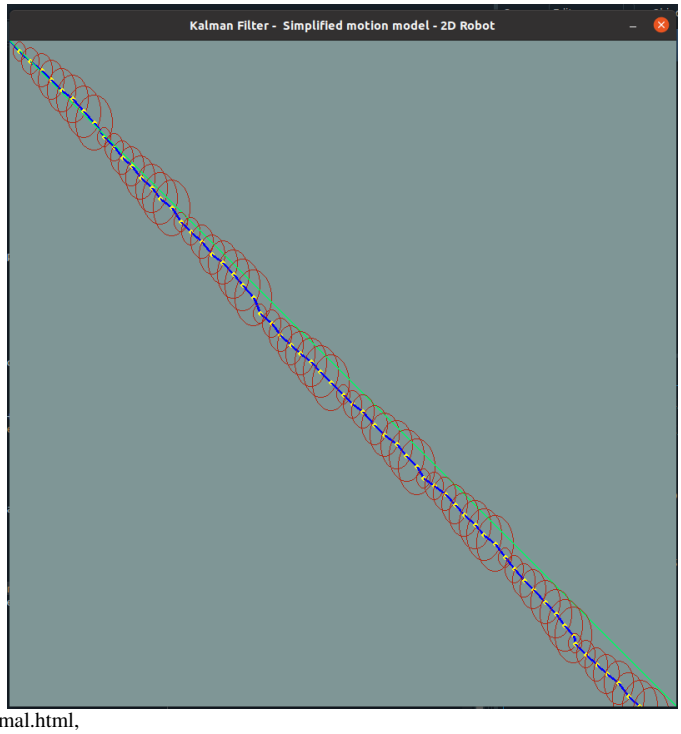

Fig. 3. Estimated 2D pose, considering the given values are std.