Kalman Filter for Object Tracking

Fahad Mirza (1001003682), University of Texas at Arlington

***Abstract*— In this report I describe how to track, estimate and predict an object using kalman filter. For this experiment a line following robot was used, which follows a black line on a white surface. Basic image processing technique was used to measure the position of the robot on the image. Matlab toolbox was used for image processing.**

***Index Terms*—Image processing, robot, kalman filter, matlab.**

I. Introduction

KALMAN filter, named after R. E. Kalman, was developed to solve discrete-data linear filtering problem. It is a recursive algorithm, represent by couple of mathematical equations, which provides better estimation and prediction of system state than single measurement system. Although the algorithm incorporates statistical noises and inaccuracies, the estimation converges to the system model trajectory over time, makes the technique very robust. It even works when the system model is not known.

There are numerous applications for Kalman filter. It has been the subject of extensive research and application in the field of robotics (for motion planning and control, trajectory optimization), autonomous and assistive navigation and signal processing. The algorithm works in two process. Prediction step and measurement step. In prediction step it predict the current state(s) including uncertainties. Then in measurement step the estimation gets corrected.

The Kalman filter addresses the general problem of trying to estimate the state of a discrete-time controlled process that is governed by the linear stochastic difference equation.

$$x_{k+1} = Ax_k + BU_k + w_k \ldots\ldots\ldots (1)$$

with a measurement,

$$z_k = Cx_k + v_k \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Where,

$x \in R^n$ = system state
U = control input
w = process noise
v = measurement noise

The noises are assumed to be independent (of each other), zero mean and with normal probability distributions,

$$p(w) \sim N(0, Q)$$

$$p(v) \sim N(0, R)$$

where Q and R are process noise covariance and measurement noise covariance, respectively. The equations are as follows,

Time Update

Estimation, $\hat{x}_{k+1}^- = A\hat{x}_k + BU_k$

Error Covariance, $P_{k+1}^- = AP_kA' + Q$

Measurement Update

Kalman Gain, $K = P_{k+1}^- C'(CP_{k+1}^- C' + R)^{-1}$

Error Covariance, $P_{k+1} = (I - KC)P_{k+1}^-$

State Estimation, $\hat{x}_{k+1} = \hat{x}_{k+1}^- + K(z_{k+1} - C\hat{x}_{k+1}^-)$

The rest of the report is organized as follows: in Sec. II I presented 1D implementation of Kalman filter, tracking only one axis (e.g. x-point). For 2D implementation (x and y point), measurement part was done by image processing method. Sec. III covers that. Sec. IV consists of the actual Kalman filter implementation. Finally, conclusion and future work is presented in Sec. V.

II. 1D Kalman Implementation

Let's assume we are going to track a moving object which follows Newton's Law of motion and we like to track position and speed. So we can write,

Position, $P_k = P_{k-1} + V_{k-1}t + \frac{1}{2}a_kt^2$

Velocity, $V_k = V_{k-1} + a_kt$

If we rearrange these equations in a state form (equ (1)), then

we have,

$$\hat{x}_{k+1} = \begin{bmatrix} P \\ V \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_{k-1} \\ V_{k-1} \end{bmatrix} + \begin{bmatrix} T^2/2 \\ T \end{bmatrix} a_k + Q$$

As for measurement, we can write equation (2) as,

$$[P] = [1 \quad 0] \begin{bmatrix} P_{k-1} \\ V_{k-1} \end{bmatrix} + R$$

R and Q are covariance matrix for measurement and process, respectively and can be written as,

$$Q = \begin{bmatrix} \sigma_p^2 & \sigma_p \sigma_v \\ \sigma_v \sigma_p & \sigma_v^2 \end{bmatrix}$$

where σ_p is position variance and σ_v is velocity variance. This system will take acceleration as control input and the noise will be added to this control input. So the effect of acceleration noise on these variances will be,

$$\sigma_p = \frac{T^2}{2} \times AccNoise$$

$$\sigma_v = T \times AccNoise$$

As we are only measuring position so we will choose a constant offset as a variance for measurement.

$$R = \sigma^2$$

Result:

Let's assume measurement system has high standard deviation. So the response will be look like:

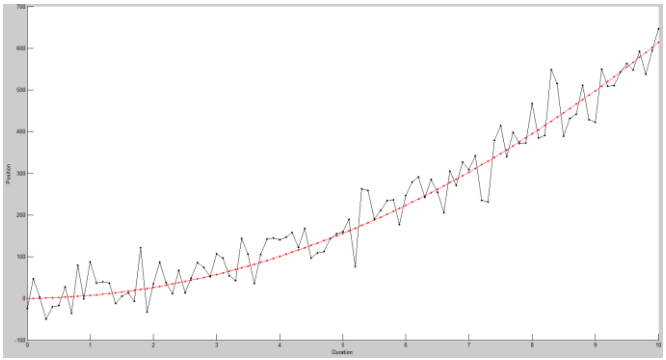

Fig. 1. Actual (R) and measured (B) position Vs time

The red line represents the object's actual position whereas the black line is the measured position of the object. If we use moving average with a window size of 5 to mitigate the error, we get,

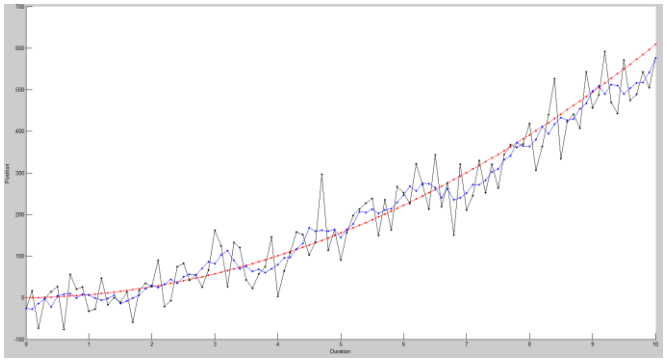

Fig. 2. Moving average output

The outcomes are better than the measured value but not good enough. The results can be improved by increasing the window size but, in nature moving average is computationally heavy and might not a good fit for real-time systems. If we use Kalman filter we get,

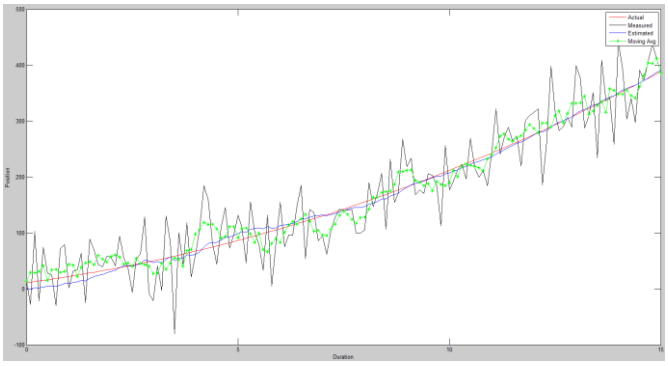

Fig. 3. Estimated output using Kalman Filter

The blue line represents the estimated position and it closely follows the actual path, unlike moving average (green).

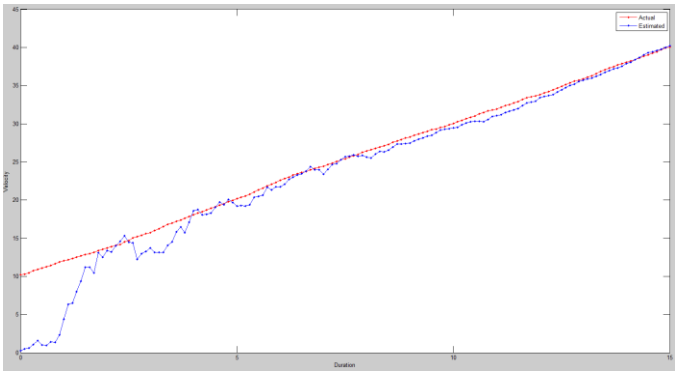

Fig. 4. Velocity Vs Time

The estimated velocity started from zero but quickly catch up to the actual value. The figure below shows the change of covariance (P) over time.

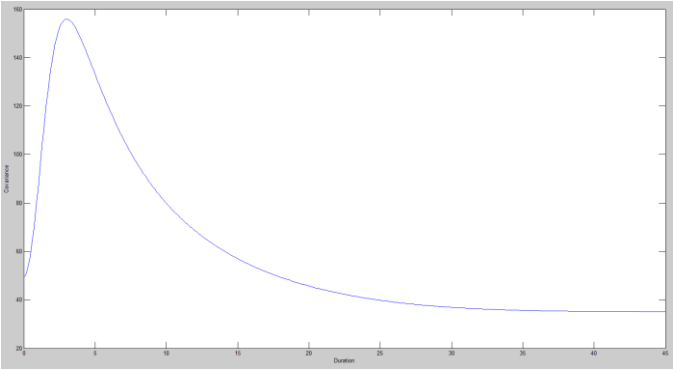

Fig. 5. Covariance Vs Time

In the next section a description of the image processing technique will be explained. This is used to track a moving object, more precisely, a line following robot, from a video and saved that as a measured position (x and y) of the robot.

III. IMAGE PROCESSING

A line following robot will follow a black line on a white surface. The video of the moving robot is then converted into images and then the robot's position will be extract from those. We know all the available colors are combination of three colors, Red, Green and Blue (RGB). Likewise all the pixels of an image are combination of RGB with a value range between 0-255. So if we subtract the background then we can separate the feature (which is the robot in this case) from the image.

This technique works for static background. So during the video recording the camera doesn't move in order to keep the background same. At the beginning of the video we collect the background images without the robot. We then take an average values of the pixels of first couple of frames to create a smooth template. We really don't care of the RGB values, so during the simulation only one dimension was used. Once the robot start to move, by subtracting the background (i.e. static color values) we can easily extract the robot's nominal position.

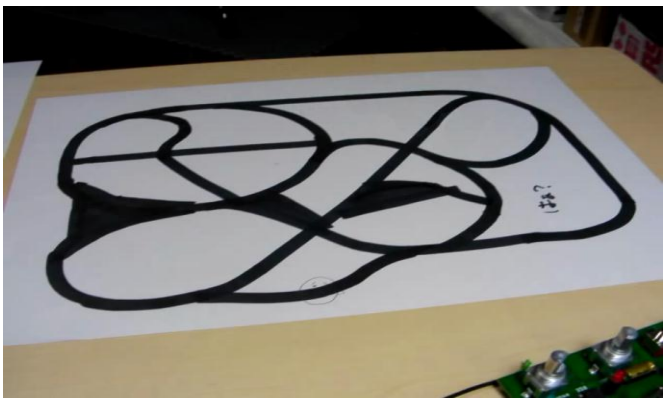

Fig. 6. The background

Figure 7 shows the raw image after taking consideration of one dimension instead of all three (RGB).

Fig. 7. Raw image taking only one dimension

Fig. 8. Raw image with robot in it

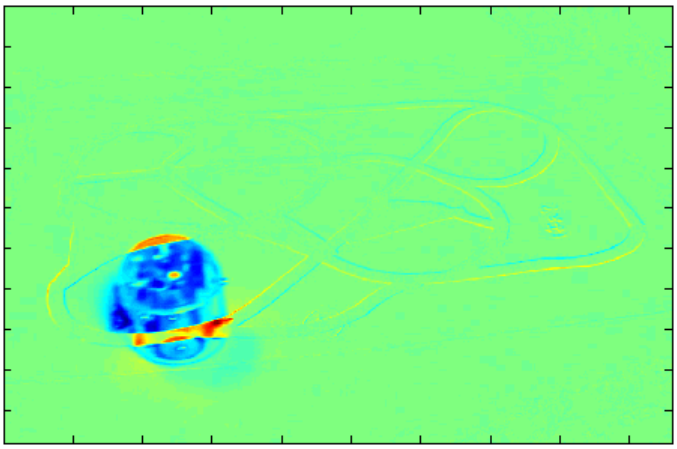

Fig. 9. After subtracting background from Fig. 8

Figure 8 shows an image with robot in it. And figure 9 shows how it looks after subtract the background (Fig 7). Now to smooth and soften the image appropriately a Gaussian filter was used. The process convolves the filter function over the image pixel by pixel. This will get rid of any white noises that is present on the image. Figure 10 shows the meshplot of the Gaussian filter. Applying filter changes figure 9 into figure 11.

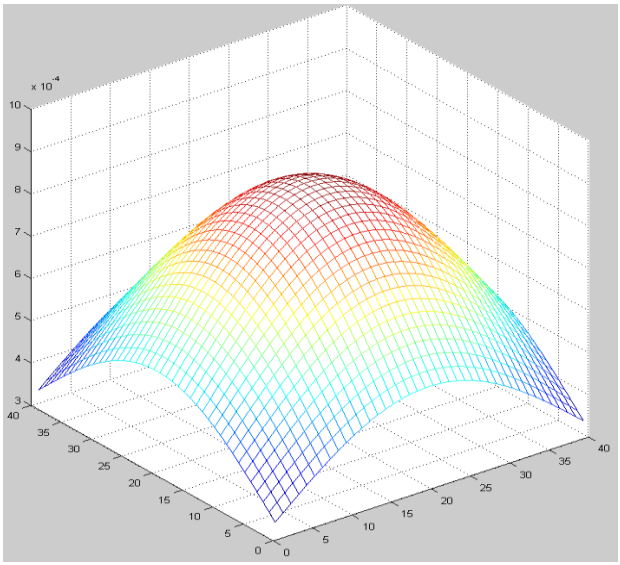

Fig. 10. Gaussian Filter Function

Now to find the object's position we have to threshold the image. We are going to use histogram based thresholding. If we use 10 equally spaced containers then we get figure 12. Histogram result shows that anything less than -20 should the robot. We don't the whole robot's structure in order to track instead we will find the center of mass of the robot. So a middle value between -20 and -40 was chosen. We will set as zero that is greater than -35 and one if smaller. That will give us figure 13.

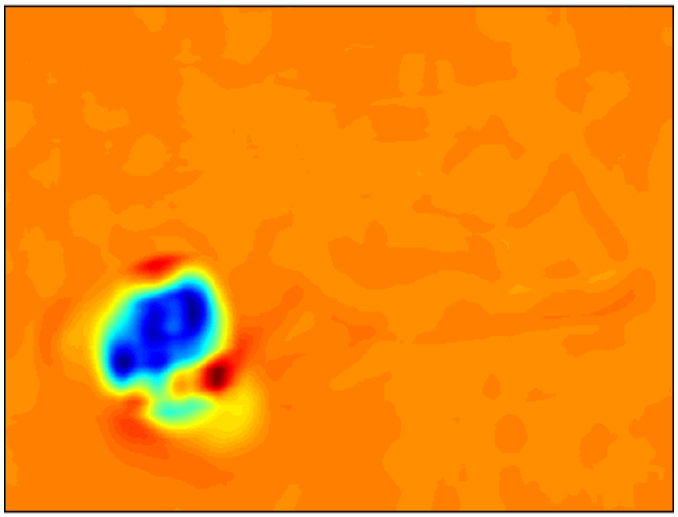

Fig. 11. After applying Gaussian filter

The robots coordinate, x and y, was evaluated as a pixel position. So center of mass is calculated as the average values of one's position (fig. 13) in the array. Figure 14 shows the center of mass of figure 13. Check out the video.

IV. 2D Kalman Implementation

Like 1D implementation equations, here too I used Newton's Law of Motion for the robot. But this time we will have equations for both x and y direction.

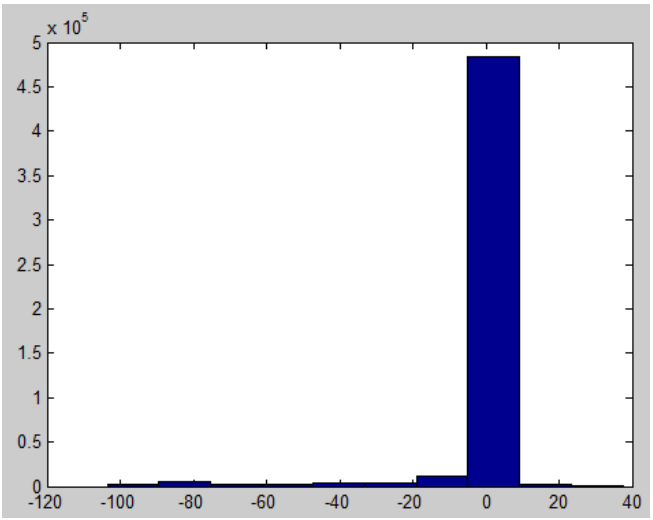

Fig. 12. Histogram of figure 11

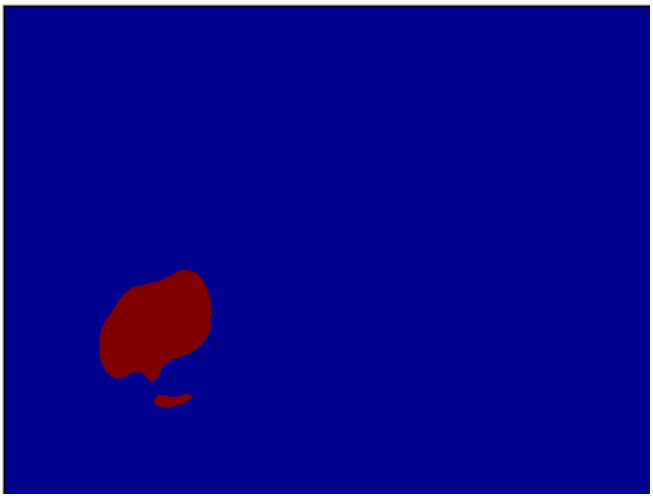

Fig. 13. Thresholding figure 11

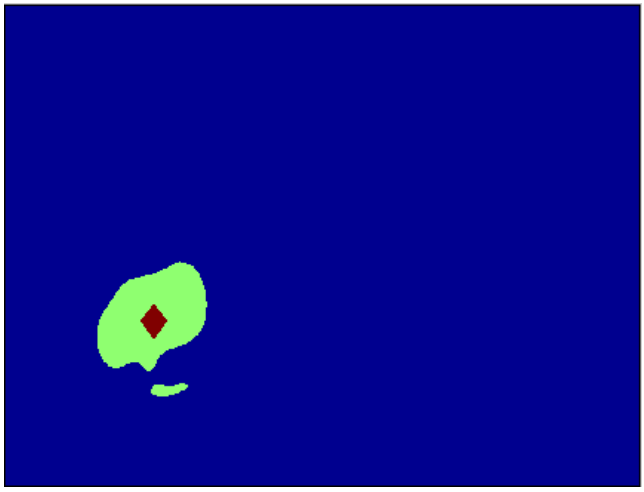

Fig. 14. Center of mass of figure 13

$$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{T^2}{2} \\ \frac{T^2}{2} \\ T \\ T \end{bmatrix} a_k + Q$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \end{bmatrix} + R$$

$$R = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}$$

$$Q = \begin{bmatrix} \frac{T^4}{4} & 0 & \frac{T^3}{2} & 0 \\ 0 & \frac{T^4}{4} & 0 & \frac{T^3}{2} \\ \frac{T^3}{2} & 0 & T^2 & 0 \\ 0 & \frac{T^3}{2} & 0 & T^2 \end{bmatrix} \times AccNoise$$

There is no direct relation between x and y so any kind of product between $\sigma_{x \text{ and }} \sigma_y$ equals to zero. The data collected from the image processing is saved in a mat file and later used in kalman filter. There were two situations that were deliberate in order to see how Kalman filter handle those situations. For a brief moment of time there wasn't any measurement data. But Kalman filter still followed the robot, because it knows the robots model.

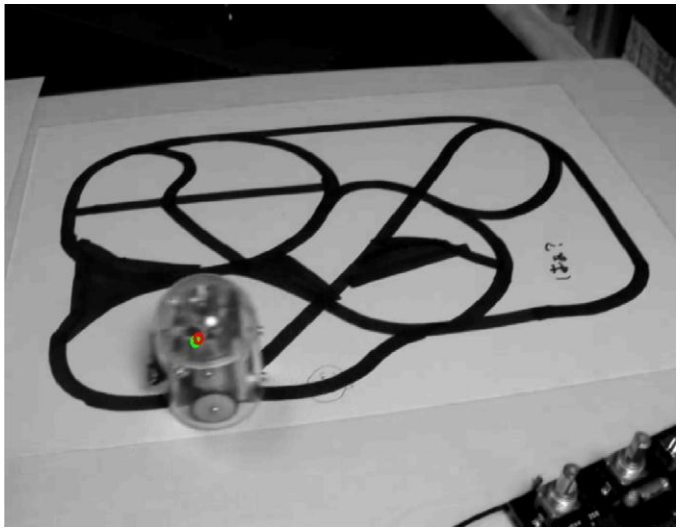

Fig. 15. Tracking the robot with Kalman filtering

The green circle represents the measured data from image processing and the red circle represents estimated data. Figure 16 shows Kalman filter can keep track of the robot in the event of no measurement.

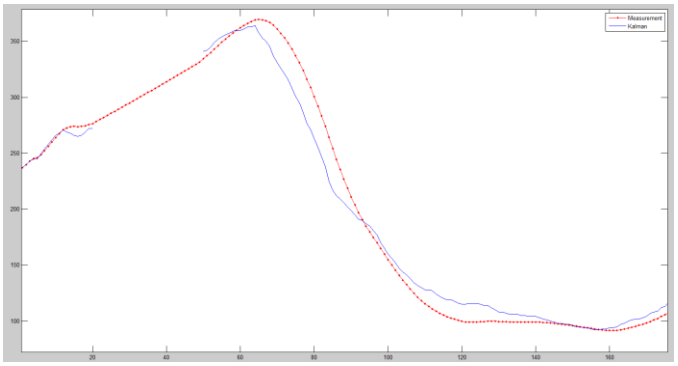

Fig. 16. X-axis data with missing measurement

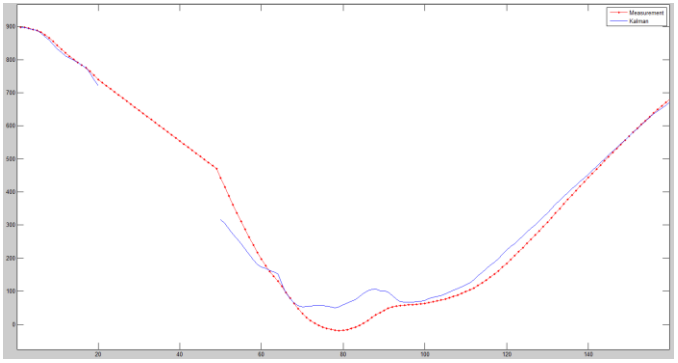

Fig. 17. Y-axis data with missing measurement

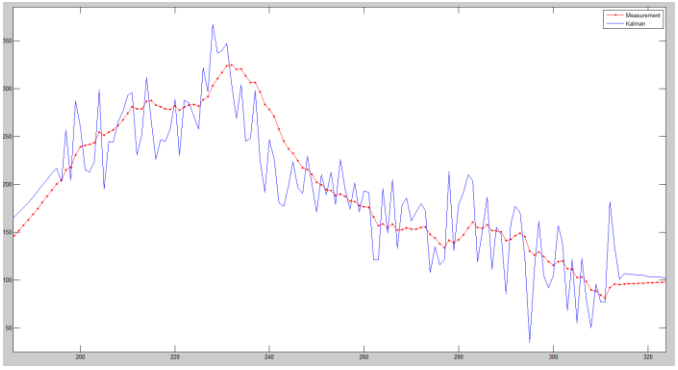

Fig. 18. X-axis data with noise added

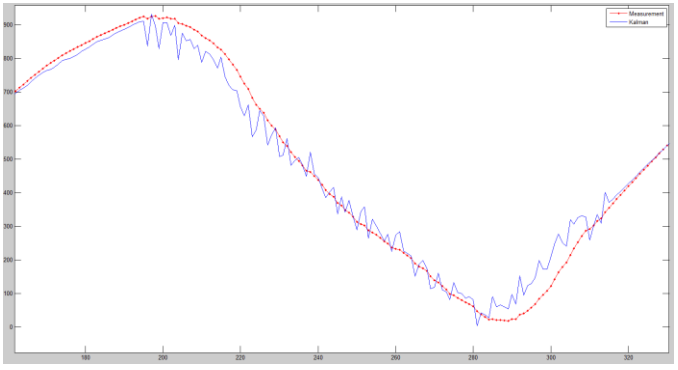

Fig. 19. Y-axis data with noise added

Figure 18 and 19 shows how well Kalman filter handled the data in the event of high noise.

V. Conclusion

In this repost I discussed how can we use Kalman filter to track a moving object. With the help of basic image processing technique a line follower robot was tracked. Deliberately two situations was introduced to check the performance of the Kalman filter and the results are shown in previous section. Newton's Law of Motion was used for the robot which is not practical for real world system. For example, during the time, when the measurement wasn't available, if then the robot would have taken a turn Kalman filter wouldn't be able to track that because there wasn't any parameter in the model that take account of the rotation. But for the sake of simplicity I have kept it that way.

Considering future work, there are lots of things that can be done to carry on forward. For instance we can start with a better system model for the robot. Then we can work on to figure out the actual noise covariance. Also we can extend this with a dynamic background.

References

[1] "Kalman Filter for Object Tracking", [Online]. Available: https://www.youtube.com/watch?v=wq69iuFC0OE. [Accessed 12 December 2015].