The Kalman Filter

Nick Watson
Electrical and Computer Engineering
University of Arizona
Tucson, USA
nicholaswatson@email.arizona.edu

Austin Pierson
Electrical and Computer Engineering
University of Arizona
Tucson, USA
apierson5@email.arizona.edu

I. Abstract

Kalman filtering is a common signal processing algorithm that is used to provide estimates of unknown values when provided with measurements over a period of time while still considering the noise of a system. This paper will provide information regarding the theory behind and the implementation of several Kalman filters. Beginning with a brief introduction outlining the history of the Kalman filter, and then moving on to a description of the theory and math behind the algorithm used in the filtering process for both the regular and extended Kalman filters. After covering the theory and equations involved, this paper will then show an implementation of the process in both one and two dimensions using code written in Python. The implementation section will also feature figures and a video which show the effectiveness of the Kalman filter, as well as a discussion on how changing various parameters in the algorithm will affect the accuracy of the calculations. Finally, this paper includes some brief examples of applications for Kalman filters and conclusions that were formed from the research performed. This paper is being written as a requirement for ECE 529 Digital Signal Processing at the University of Arizona.

II. Introduction

The Kalman Filter was developed by Rudolf Emil Kalman in 1960. Kalman published a paper which explained a recursive solution to a discrete data linear filtering problem. Since this paper was published, the Kalman filter has been developed which has led to multiple advances in the digital computing field. The Kalman filter is an algorithm that analyzes different measurements over an amount of time. Many times, these measurements will have noise or other factors that might affect the accuracy of the measurements. The Kalman filter is able to analyze these measurements, and while still accounting for inaccuracies in the signal, produce an estimate for the next measurements that will be recorded.

Originally developed in 1960 to help develop a navigation error with the Apollo project, the Kalman filter still has many applications today. Most of these applications have been in fields related to navigation or robotic motion tracking, particularly in the aerospace field. Throughout this paper we will be analyzing two types of filters. The normal Kalman filter which is used to estimate the state of linear systems, and the extended Kalman filter which is used to estimate the state of discrete-time nonlinear systems. We will begin by first explaining the algorithm used to design a Kalman filter, then we will show our implementation of a normal and extended Kalman filter using python, and finally we will discuss the differences in the implementation of the two types of Kalman filters and what we've learned in our tests.

III. Kalman Filter Algorithm

A. Basic Kalman Filter

In this section we will discuss the algorithm behind the Kalman filter, this will be done by working through the six major equations used to create this algorithm. To begin, the Kalman filter is better described as an estimator then it is as a filter. Without involving state matrices, the most basic equation describing the Kalman filter is as follows

$$\hat{x}_k = K_k \cdot Z_k + \left(1 - K_K\right) \cdot \hat{x}_{k-1}$$

In this equation the X_k is the current estimation of the state of the object that is being tracked. The K_k is the Kalman filter gain, the Z_k is the measured value, and the X_{k-1} is the previous state's estimated values. The subscript k's in the equation are meant to represent the incrementation of different states. For our project these k values are likely to be increments of time.

The first step to using the Kalman filter is checking if

$$x_k = Fx_{k-1} + Gu_k + w_{k-1}$$

the system in question can be represented by the Kalman filter parameters. The two equations to check this are as follows

$$z_k = Hx_k + v_k$$

By looking at the first equation, we can learn that the signal, x_k can be determined from a linear, randomly determined equation. We can also see that the signal, x_k can be shown as a linear combination of a previous value added together with the process noise. Generally, in most cases, there is not a control signal, u_k. [2] In this paper we will be assuming that u_k is zero. The second equation, which describes the measurement value z_k, tells us that this measurement value is gaussian, and a linear combination of the signal value and the measurement noise, v_k. In most cases, the values F, G, and H will be matrices, and while occasionally these matrices might change between states, they generally remain constant. For the purpose of this

$$\mathbf{F} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix}$$

paper these are the values of matrices we will be using.

The Kalman filter is broken down into two sets of equations, the time update or prediction step, and the measurement update or correction step. The first two equations we will be using are a part of the prediction step.

Table 1: Kalman Filter Prediction Equations

Kalman Filter Prediction Step Equations
$\hat{x}_k^- = F\hat{x}_{k-1} + Gu_k$
$P_k^- = F \cdot P_{k-1} \cdot F^T + Q$

[2]In this prediction step, the Kalman filter estimates the current value of the state variables, including their uncertainties. The predicted state created from the previous state estimate. The new term, P, is referred to as the state error covariance, this variable keeps track of the error that the filter thinks the estimate error has. Covariance is calculated by measuring the joint probability of two random variables. You can see that during the prediction step, the error is highest because of the summation of Q, which is a process noise variable/matrix. Q is found by a multiplication between G, G transposed, and the acceleration variance.

$$\mathbf{Q} = \mathbf{G}\mathbf{G}^{\mathrm{T}}\sigma_a^2 = \begin{bmatrix} \frac{1}{4}\Delta t^4 & \frac{1}{2}\Delta t^3 \\ \frac{1}{2}\Delta t^3 & \Delta t^2 \end{bmatrix}\sigma_a^2.$$

The next step in Kalman filtering is the measurement update or correction step, this phase is shown in these four equations.

Table 2: Kalman Filter Update Equations

Kalman Filter Measurement Update Step Equations
$K_k = P_k^- \cdot H^T \cdot \left(HP_k^- H^T + R\right)^{-1}$
$\hat{x}_k = \hat{x}_k + K_k\left(z_k - H\hat{x}_k\right)$
$P_k = \left(1 - K_k H\right)P_k^-$
$\hat{y}_k = z_k - Hx_k^-$

[2]In the update stage the first value that is calculated is y^-_k, this value is the measurement residual. Or in other words, is the difference between the true measurement, z_k and the estimated measurement, Hx_k^-. The filter estimates the current measurement by multiplying the predicted state by the measurement matrix. The measurement residual is multiplied by the kalman gain to account for the correction to the predicted estimate, x^-_k. After the updated state estimate is calculated, the Kalman filter then updates the predicted error covariance which was mentioned in the prediction step of the process, this value is represented by P_k. This value will then be used in the following time step.

After gathering all of the equations necessary for both the prediction and measurement update steps, we can begin the Kalman filtering process. The Kalman filter is able to create a prediction for the state of the system by averaging all of the predictions about the state and taking into account the measurement data. The Kalman filter also uses the concept of Kalman gain, represented in our set of equations by K, which is a value between 0 and 1. This value determines the importance of the measurements that have been recorded and the estimation of the current state. A high Kalman gain value corresponds to a high weighting on the most recent measurements, if the gain is closer to 0 the system will have a higher weighting on the prediction values. When visualizing the concept of Kalman gain with a graph, a gain close to 0 will be a very smooth line that is not responsive to noise, while a gain closer to 1 will be a much more responsive line that appears to jump around. The Kalman gain can be tuned to fit the requirements of the system.

Here is the process that is repeated during the Kalman filtering process [3]

1. The system receives a measurement from the object relative to a starting position
2. The system initializes the object based on the data from the first step
3. A time period is established, Δt, after this time period another measurement is taken
4. The system then performs the prediction step, where the next position is calculated by using velocity * Δt. Many times the velocity will be

constant, especially when working with the extended Kalman filter

5. The system then compares this calculation with the measured data, this is the measurement update step, in this step the system takes into account the Kalman gain and determines how much to weight the predicted values vs. the measured values
6. This process is repeated, predicting and updating the position of the object

B. 3.2 Extended Kalman Filter Algorithm

The second type of Kalman filter that we will be discussing in this paper is the Extended Kalman filter. The Extended Kalman filter is an extension of the original filter, but this time working with non-linear systems and approximating this nonlinearity by using first and/or second order derivatives. A Kalman filter that linearizes about the current mean and covariance is referred to as an extended Kalman filter. The normal Kalman filter uses a set of equations that are executed recursively, only working with states that are first order. This is because an assumption is being made that the higher order effects are so small that they can be ignored when predicting a position. The Extended Kalman filter will be taking into account these effects.

[1]The difference between the normal Kalman filter and the Extended Kalman filter is the use of a Jacobian matrix. Similar to Taylor Series, we are able to linearize the estimation around a current estimate through the use of partial derivatives of our process and measurement functions, we use the Jacobian matrix for this linearization. The Jacobian matrix of a vector-valued function in several variables is the matrix of all its first-order partial derivatives. It is defined as

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Similar to the original Kalman filter, the extended version still uses a state vector x, but this time the state vector is a non-linear difference equation. Here are the new non-linear equations which are used for the state vector, x, and measurement, z.

$$x_k = f\left(x_{k-1}, u_k, w_{k-1}\right)$$

$$z_k = h\left(x_k, v_k\right)$$

Once again, the variables defined as w_k and v_k will be used to show the process and measurement noise. The non-linear function h relates the state x_k and the measurement z_k. Generally when beginning the filtering algorithm, one does not know the noise w_k and v_k but the state and measurement vectors can still be approximated without noise, by setting these values to 0.

$$\hat{x}_k = f\left(x_{k-1}, u_k, 0\right)$$

$$\hat{z}_k = h\left(x_k, 0\right)$$

With these four equations shown above, we are able to create governing equations to estimate a process with non-linear difference and measurement relationships through the use of linearization.

$$x_k = \hat{x}_k + A\left(x_{k-1} - \hat{x}_{k-1}\right) + Ww_{k-1}$$

$$z_k = \hat{z}_k + H\left(x_k - \hat{x}_k\right) + Vv_k$$

There are four Jacobian matrices which are used in these equations:[1]

- A is the Jacobian matrix of partial derivatives of f with respect to x
- W is the Jacobian matrix of partial derivatives of f with respect to w
- H is the Jacobian matrix of partial derivatives of h with respect to x
- V is the Jacobian matrix of partial derivatives of h with respect to v

The next step in Extended Kalman filtering is determining the prediction and measurement errors, with these values we are able to determine whether the filter should weigh the predicted or measurement values higher.

Prediction error equation:

$$\hat{e}_{xk} = x_k - \hat{x}_k$$

Measurement error equation:

$$\hat{e}_{zk} = z_k - \hat{z}_k$$

We also are able to use the measurement error to find the error of the system using the Kalman gain as well,

$$\hat{e}_k = K_k \hat{e}_{zk}$$

[5]You can see that similar to the normal Kalman filter, the Kalman gain value, K_k, is a ratio of error values to determine how much to weight the prediction values versus the measurement values. Finally, we must establish the Extended Kalman filter measurement and update equations.

Table 3. EKF Prediction Step Equations

Extended Kalman Filter Prediction Step Equations
$\hat{x}_k^- = f\left(\hat{x}_{k-1}, u_k, 0\right)$
$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$

Table 4. EKF Measurement Update Step Equations

Extended Kalman Filter Measurement Update Step Equations
$K_k = P_k^- H^T \left(H_k P_k^- H_k^T + V_k R_k V_k^T\right)^{-1}$
$\hat{x}_k = \hat{x}_k^- + K_k\left(z_k - h\left(\hat{x}_k^-, 0\right)\right)$
$P_k = \left(1 - K_k H_k\right) P_k^-$

As covered in section 3.1, the prediction process is composed of two steps, the first being predict the state ahead, this is represented by $\widehat{x_k^-}$ and then predict the error covariance ahead which is represented by P_k^-. The measurement update process is then performed in three steps, first compute the Kalman gain, K_k , the update the estimate with the measurement z_k this is $\widehat{x_k}$ and then finally, update the error covariance, which is done in the equation for P_k.

IV. Kalman Filter Implementation

In our project we will be implementing the normal Kalman filter in one and two dimensions. We will begin by discussing how the basic one dimensional Kalman filter is implemented in Python. We will discuss the two dimensional implementation later in this section.

A. One Dimensional Kalman Filter

As mentioned in the introduction there are two steps to Kalman filtering. To implement the Kalman filter in Python, we need to apply the equations in Table 1 and Table 2. To apply these equations, however, there are some initial steps that need to be taken.

The first of these steps is to establish the initial position and initial velocity along with the acceleration variance. The initial position value will be initialized to 0, the initial velocity value will be set to 1. The final value that needs to be set is the acceleration variance, ideally this value is close to 0. This is because this value measures how far a set of numbers is spread out from their average value. Setting the acceleration variance at a value close to 0 allows us to have greater accuracy in our filtering algorithm. We will discuss how changing the acceleration variance value affects the performance of our Kalman Filter. The final value that needs to be initialized is Δt, which will determine the frequency of our measurements. We will set this variable to 0.1 to frequently measure the data. We have set our program to predict the position 100 times and we will be performing an update every 5 predictions.

For the first implementation we will work with a line with a constant velocity of 0.9. The position will have a standard deviation of 0.1 (this will be important for whenever the filter is updated). Below is a simulation of the one dimensional Kalman filter with two different values for acceleration variance, a low value of 0.2 and a high value of 2.

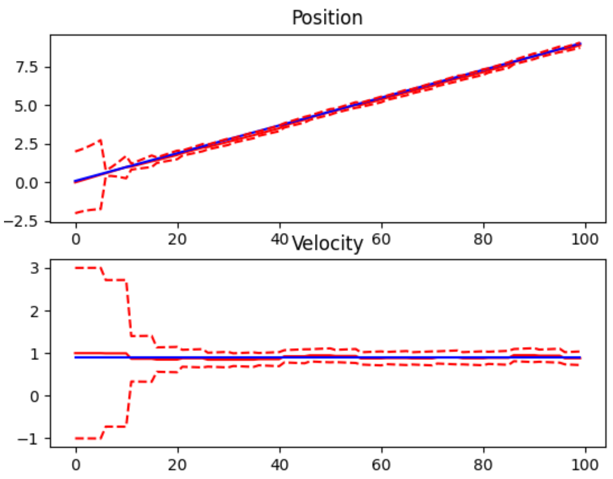

Figure 1. Low Acceleration Variance Plots

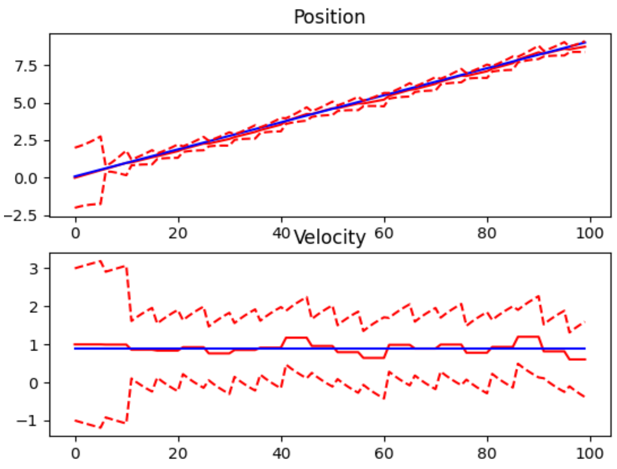

Figure 2. High Acceleration Variance Plots

In Figures 1 and 2 the dotted red lines are the Kalman filters' first (+/-) standard deviations. The solid red lines are the actual values of the Kalman filter. The blue lines are the real data that served as the basis for making predictions. The two plots in Figure 1 were filtered using an acceleration variance of 0.2. The two plots in Figure 2 were filtered using an acceleration variance of 2.0. Other than the acceleration variance, all other parameters as well as the data used were identical. You can see that every 5 steps, the Kalman filter performs its measurement correction update and the red lines (solid and dotted) converge towards the solid blue lines. The effect of the acceleration variance becomes strikingly obvious when observing the velocity plots of the two figures. At first, when there is little data about the system, the standard deviations (dotted red lines) are very far away from the actual values. As more predictions are made and more data is observed, the standard deviations get closer to the actual values. This is a lot more obvious in Figure 1, where around the 20th prediction that is made the standard deviation is very close to the real data. Additionally, the Kalman filter's predicted velocity is almost identical to the actual velocity. For Figure 2, while the standard deviation of the velocity gets slightly closer to the actual value, it is never as accurate/representative of the actual data. In addition, there are several points where the Kalman filter's velocity value (solid red line) is significantly different from the actual value. You can see that for the position plots the behavior is very similar to that of the velocity but not as dramatic/obvious. In short, lower variance is desired for the Kalman filter to be accurate.

B. Two Dimensional Kalman Filter

Up to this point, we have only worked with a one dimensional Kalman filter. It should be noted that it is very easy to turn the Kalman filter into a two dimensional system. The prediction and measurement correction step equations are exactly alike. However, the size of the matrices must be increased to account for the measurement and prediction in both the x and y directions. Let's start with the state vector:

$$\bar{x} = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

The state vector represents what we will be keeping track of in a 2-D (or 4-D in terms of matrices) Kalman filter: two positions and two velocities. It is important to note that the measurement matrix, z, must have the same form as the state vector (i.e. it must be a 4x1 matrix in this case). The orientation of the state vector is also important because it is used to determine the form of the state transition matrix, F, below:

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Recall that the F matrix is used in the prediction step of the Kalman filter (see Table 1). Another matrix that needs to be changed is the observation matrix, H.

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

The H matrix is needed to "observe" the changes in the positions and velocities. Depending on how the Kalman filter is implemented, however, the H matrix could change. For instance, it may not necessarily need to observe changes in velocities. In which case, when calculations are performed in the update step of the algorithm the velocities wouldn't be modified. The H matrix above is the default matrix used for observing both positions and velocities.

The process covariance matrix, P, also changes in size to a 4x4 matrix. It should be noted that this matrix must always match the number of values in the state vector. If there are four values in the state vector, then there must be four rows and four columns in the P matrix.

The final matrices of note are the noise covariance matrices Q and R. These are both 4x4 matrices. The Q matrix is the process noise covariance matrix, while the R matrix is the sensor noise covariance matrix. Below is the representation of the Q and R matrix.

$$Q,\ R = \begin{bmatrix} \sigma_x^2 & \sigma_{x\,y} & \sigma_{x\,\dot{x}} & \sigma_{x\,\dot{y}} \\ \sigma_{y\,x} & \sigma_y^2 & \sigma_{y\,\dot{x}} & \sigma_{y\,\dot{y}} \\ \sigma_{\dot{x}\,x} & \sigma_{\dot{x}\,y} & \sigma_{\dot{x}}^2 & \sigma_{\dot{x}\,\dot{y}} \\ \sigma_{\dot{y}\,x} & \sigma_{\dot{y}\,y} & \sigma_{\dot{y}\,\dot{x}} & \sigma_{\dot{y}}^2 \end{bmatrix}$$

In real-world Kalman filter applications, the R matrix is given by a sensor's sensitivity. As a result, the values that make up this matrix can typically be determined by a manufacturer. Otherwise, R can be taken as an identity matrix that is multiplied by a scalar with a magnitude between 0 and 1. It should be noted that the Q matrix is obtained from noise levels in a system at steady state. If there is little noise in a system the Q matrix will essentially be a zero matrix.

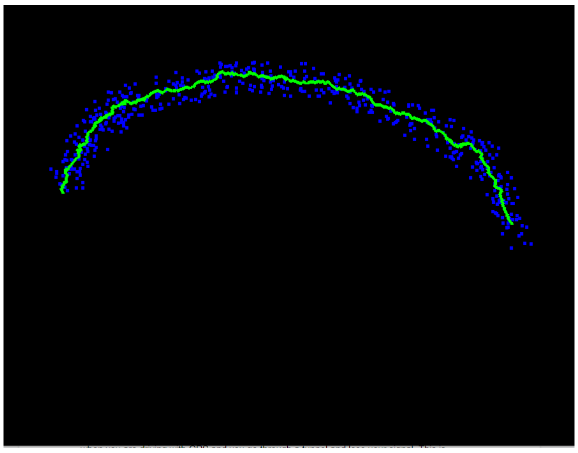

Figure 3. 2D Kalman Filter with a noise of 50

Figure 3. shows an example of our implementation of a 2-D Kalman filter. The implementation was developed in Python, and tracks a cursor's movement using the Kalman filter. The matrices discussed in this section were used along with Table 1 and Table 2 to develop the algorithm. The important thing to note about the implementation is that we set the values inside the Q and R matrices to be very small. The R matrix was given a scalar of 0.2, and the Q matrix was given a variance of 0.001 for the x and y velocities. The reason for this is because we wanted to use this tool to observe the effect of noise on the Kalman filter. Having high values for these matrices means that we don't trust the way that we are recording our measurements, and the noise filtering properties of the Kalman filter algorithm will get weaker. We took exact values for positional noise (which were randomly generated within a certain range that can be set within the implementation) and added them to the x and y positions of the cursor whenever we applied the Kalman filter algorithm. The green line represents the result of the update process in the Kalman filter's algorithm, and the blue dots represents the cursor's positions.

(**Note:** A demonstration of the implementation is given in the video provided with this submission.)

V. Applications of Kalman Filters

Within the signal processing community, Kalman filtering is a very popular and active topic. The Kalman filter is a powerful algorithm that is useful at controlling noisy systems. There are many applications for this, the most common of these are tracking objects, economics, navigation, and some computer vision applications such as stabilizing depth measurements and analyzing data from radars or laser scanners. A common example that we see in life everyday is when you are driving with GPS and you go through a tunnel and lose your signal. This is because the GPS uses the Doppler effect to provide a measurement of your speed. The Kalman filter is able to use the previous measurements of position and velocity and make accurate predictions of your next positions while filtering out noise related to the measured values, even while the GPS in your car is unable to connect with a satellite.[4]

An Extended Kalman filter could be applied to moving object tracking of a 3D plane. This situation involves uncertainty in measurement information and the need to accurately predict the trajectory of the plane. Once the object motion, measurement, and noise are established, these values can be substituted into the equations mentioned in section 3.2 of this paper. The Extended Kalman filter algorithm would then be able to process the information and provide an estimate of the trajectory of the plane while filtering out the noise included with the measurements. The predictions found by the extended filter algorithm can even be enhanced by utilizing the Unscented Kalman Filter, which was not discussed in detail in this paper.

The Unscented Kalman Filter is an upgraded version of the Extended filter algorithm. The Extended Kalman filter takes one point which has an average and an approximate associated with it to form a prediction. In contrast, the Unscented filter takes multiple points, called sigma points with multiple averages and approximations. These sigma points are then assigned a weighting for the importance of the point. These weightings are calculated through an equation which takes into account the mean of the gaussian, dimensionality of the system, the scaling factor, and the covariance matrix. The combination of these multiple weighted sigma points creates even more information which leads to an even more accurate prediction formed through the Unscented Kalman filter algorithm. This form of algorithm has many applications related to technology companies looking for a way to find accurate positioning predictions when inside of buildings. This leads to the use of the Unscented Kalman filtering algorithm being very common in Inertial Measurement Units (IMU) and other navigation systems.

VI. Conclusion

Kalman filtering is a common signal processing algorithm that is used to provide estimates of unknown values when provided with measurements over a period of time while still considering the noise of a system. In this paper, we began by introducing the algorithms for both the normal and extended versions of the Kalman filter. Both algorithms have a recurring two-step cycle that involves a prediction and an update. The normal Kalman filter uses gaussian values, which implies that it can only work with nonlinear data. The Extended Kalman filter allows for nonlinear functions to be used through the use of Jacobian matrices to linearize the data.

In section IV. A we began by implementing a one dimensional normal Kalman filter. Using Python, we were able to analyze the effect of acceleration variance on the performance of the filtering algorithm. We observed that an increased acceleration variance leads to less accurate measurements resulting in poor Kalman filtering performance. In the next section we implemented a two dimensional version of the normal Kalman filter. This was done by adjusting the format of the matrices to allow for two positional measurements and two velocity measurements. We created a visual tool that allows us to observe how input noise affects the Kalman filter's output. We found that, assuming nearly ideal values of the two covariance matrices Q and R, that the Kalman filter's output follows the object's (in this case the cursor) path very closely even with higher noise values. However, when the covariance matrices Q and R are not ideal, the output of the filter did not effectively ignore the input noise in the system. Using our created visual, we were able to observe this concept by increasing the values inside the Q matrix. We then saw that our green prediction line seemed to travel to each of the measured (blue) points, this shows that the algorithm is not filtering the noise with non-ideal Q values.

In our implementation of the two dimensional Kalman filter we attempted to use the equation for noise covariance matrix Q shown in section IV B. We found that when using this formula the values in our Q matrix were too large and our Kalman algorithm did not filter the input noise as well as we would like. Our conclusion from this observation is that when the Kalman filter is implemented in 3D motion tracking there are provided noise covariance values which can be accounted for in the algorithm. These values are generally very small (this is why we chose to set our Q values to 0.001 in the simulation tool).

In conclusion, the Kalman filter is a powerful adaptive filter that is effective at filtering out statistical noise and providing accurate predictions when tracking objects. These qualities lead this concept to be popular in signal processing fields such as econometrics, guidance, and navigation.

References

[1] (www.kalmanfilter.net), Alex Becker. "Online Kalman Filter Tutorial." *Kalman Filter Tutorial*, www.kalmanfilter.net/default.aspx.

[2] Chadha, Harveen Singh. "Extended Kalman Filter: Why Do We Need an Extended Version?" *Medium*, Towards Data Science, 8 Nov. 2019, towardsdatascience.com/extended-kalman-filter-43e52b16757d.

[3] Esme, Bilgin. "Bilgin's Blog." *Bilgin's Blog | Kalman Filter For Dummies*, bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies.

[4] Kim, Youngjoo, and Hyochoong Bang. "Introduction to Kalman Filter and Its Applications." *IntechOpen*, IntechOpen, 5 Nov. 2018, www.intechopen.com/books/introduction-and-implementations-of-the-kalman-filter/introduction-to-kalman-filter-and-its-applications.

[5] Srinivasan, Sharath. "Kalman Filter: An Algorithm for Making Sense from the Insights of Various Sensors Fused Together." *Medium*, Towards Data Science, 18 Apr. 2018, towardsdatascience.com/kalman-filter-an-algorithm-for-making-sense-from-the-insights-of-various-sensors-fused-together-ddf67597f35e.