Introduction

This report details the setup and implementation of a Kalman Filter designed to filter noisy positional measurements. Two separate filters were set up and implemented to filter 2-dimensional and 3-dimensional positional measurements. The 2-dimensional input data comprised of positional measurements of a US Navy ship taken over an eight-minute time span. The 3-dimensional data input into the filter consisted of positional measurements from a US Navy submarine collected over a two-minute timespan.

2D Kalman Filter

1. Set up of the Kalman Filter

The first step of setting up any Kalman Filter is deriving a process model for the system. Since the system we are modeling a moving object, a system that models the object's position, velocity, and acceleration is the best fit. The system model for the 2-Dimensional Kalman filter can be seen in *Figure 1*. For this process model we are assuming two independent white noise inputs to the effecting the *x* and *y* components of the system.

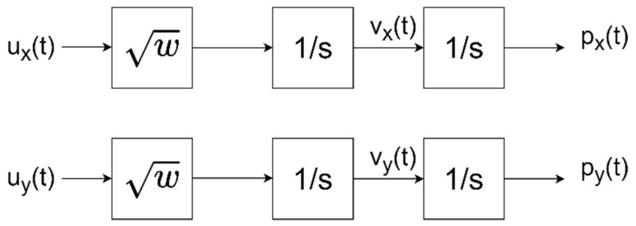


Figure 1 – 2D Kalman Filter System Model

For this system we will use a state vector ($\bar{x}$) that tracks the *x* and *y* components of both position and velocity of the object.

$$\bar{x} = \begin{bmatrix} p_x(t) \\ v_x(t) \\ p_y(t) \\ v_y(t) \end{bmatrix}$$

Since the process model is movement, we know that the derivative of position is velocity:

$$\frac{d\bar{p}}{dt} = \bar{v}$$

$$p_x'\hat{x} + p_y'\hat{y} = v_x\,\hat{x} + v_y\hat{y}$$

Using this relation, we can find the state equation for the system:

$$\bar{x}'(t) = \boldsymbol{F}\bar{x}(t) + \bar{G}u(t)$$

$$\bar{x}'(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}\bar{x}(t) + \begin{bmatrix} 0 \\ \sqrt{w} \\ 0 \\ \sqrt{w} \end{bmatrix}u(t)$$

Using the continuous state transition matrix ($\boldsymbol{F}$) we can find the discrete state transition matrix: Φ

$$\Phi = e^{\boldsymbol{F}\Delta t} = \mathcal{L}^{-1}\{(s\boldsymbol{I} - \mathbf{F})^{-1}\}|_{t=\Delta t}$$

$$(s\boldsymbol{I}-\mathbf{F})^{-1}=\begin{bmatrix} s & -1 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & -1 \\ 0 & 0 & 0 & s \end{bmatrix}^{-1}=\begin{bmatrix} 1/s & 1/s^2 & 0 & 0 \\ 0 & 1/s & 0 & 0 \\ 0 & 0 & 1/s & 1/s^2 \\ 0 & 0 & 0 & 1/s \end{bmatrix}$$

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

The proccess and noise models of of discrete sysem are represented as:

$$x_{k+1}=\phi x_k+w_k$$

$$z_k=\boldsymbol{H}_k x_k+v_k$$

Since our measurements are simply the positional components of the state vector:

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

In order to define our Kalman filter we need to find the covariance matrix of the proccess noise: $\boldsymbol{Q_k}=E\left[w_k w_k^T\right]$

$$\boldsymbol{Q}_k=\begin{bmatrix} E[p_x^2] & E[p_x v_x] & E\left[p_x p_y\right] & E\left[p_x v_y\right] \\ E[v_x p_x] & E[v_x^2] & E\left[v_x p_y\right] & E\left[v_x v_y\right] \\ E\left[p_y p_x\right] & E\left[p_y v_x\right] & E\left[p_y^2\right] & E\left[p_y v_y\right] \\ E\left[v_y p_x\right] & E\left[v_y v_x\right] & E\left[v_y p_y\right] & E\left[v_y^2\right] \end{bmatrix}$$

$$E\left[x_i x_j\right]=\int_0^{\Delta t}\int_0^{\Delta t} g_i(\mathrm{T_i})g_j(T_j)R_u(T_i-T_j)dT_i dT_j$$

Since input noise to the *x* and *y* portions of the system are indipendent and zero mean proccesses, we know that expectation of any *x*, *y* crossover term will be zero.

$$E\left[p_x p_y\right]=E[p_x]E\left[p_y\right]=0$$

Input is white noise therefore: $R_u(T_1-T_2)=\delta(T_1-T_2)$

$$E\left[x_i x_j\right]=\int_0^{\Delta t}\int_0^{\Delta t} g_i(\mathrm{T_i})g_j(T_j)\delta(T_i-T_j)dT_i dT_j=\int_0^{\Delta t} g_i(\mathrm{T_j})g_j(T_j)dT_j$$

$$g_{v_x}(t)=g_{v_y}(t)=\sqrt{w}$$

$$g_{p_X}(t)=g_{p_y}(t)=\sqrt{w}\,t$$

$$E[v_x^2]=E\left[v_y^2\right]=w\Delta t$$

$$E[p_x^2]=E\left[p_y^2\right]=\frac{w}{3}\Delta t^3$$

$$E[p_x v_x]=E[v_x p_x]=\frac{w}{2}\Delta t^2$$

$$\boldsymbol{Q}_k = \begin{bmatrix} \frac{w}{3}\Delta t^3 & \frac{w}{2}\Delta t^2 & 0 & 0 \\ \frac{w}{2}\Delta t^2 & w\Delta t & 0 & 0 \\ 0 & 0 & \frac{w}{3}\Delta t^3 & \frac{w}{2}\Delta t^2 \\ 0 & 0 & \frac{w}{2}\Delta t^2 & w\Delta t \end{bmatrix}$$

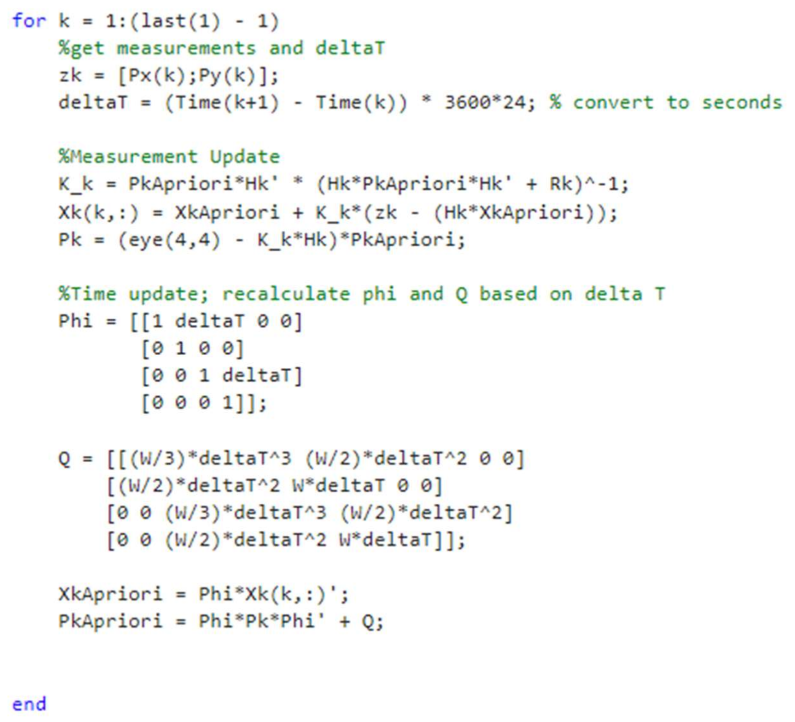
The next important step of setting up our Kalman filter involves estimating the measurement noise (v_k). For the purpose of this project, we will be assuming our measurement noise is scaled unity white noise by a factor of $\sqrt{q}$, and independent for each measurement dimension. The resulting covariance of measurement noise can be expressed as:

$$R_k = \begin{bmatrix} q\Delta t & 0 \\ 0 & q\Delta t \end{bmatrix}$$

With Φ, $\boldsymbol{H}_k$, $\boldsymbol{Q}_k$, and $R_k$found for our system we can now begin implementing our Kalman Filter.

2. Implementation of the Kalman filter

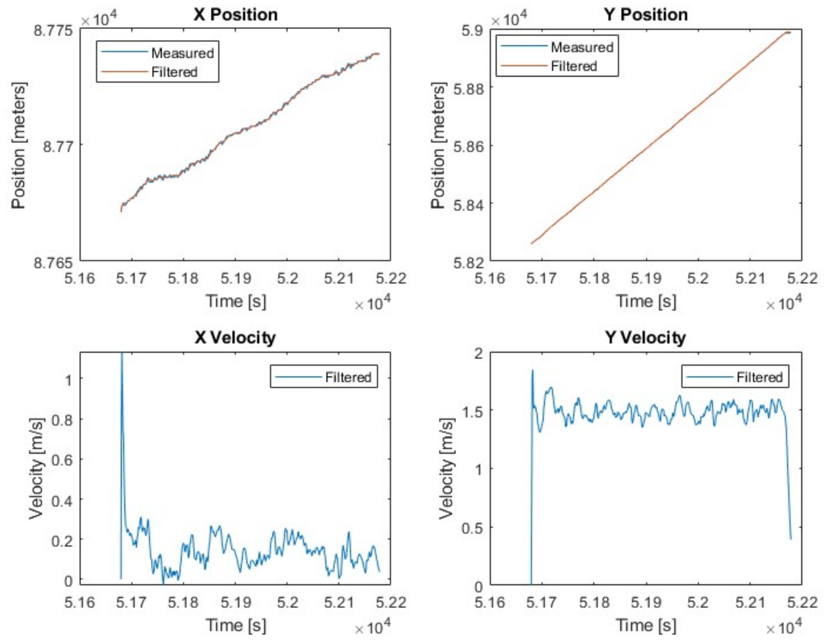
In order to implement the Kalman filter in MATLAB, the five Kalman equations were implemented with the following code:

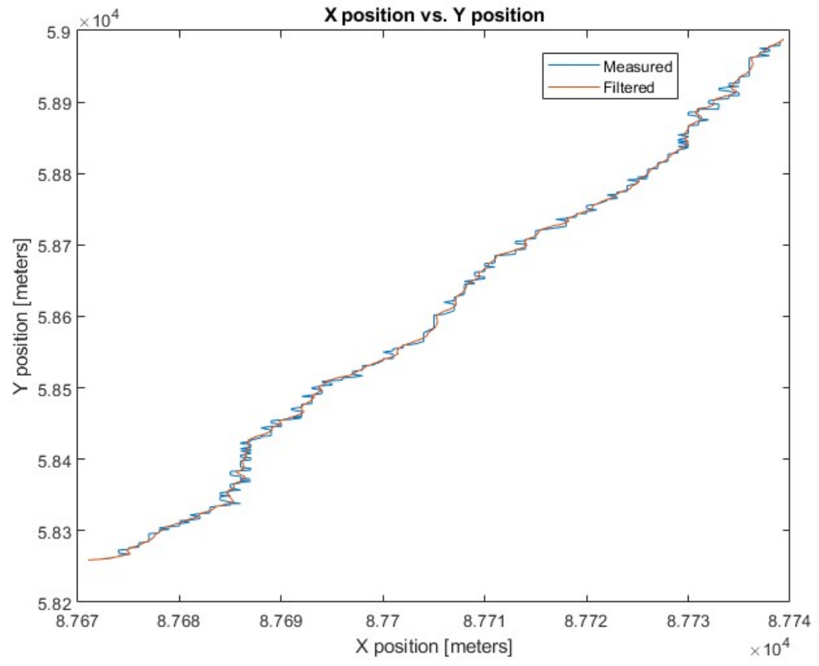


```
for k = 1:(last(1) - 1)
    %get measurements and deltaT
    zk = [Px(k);Py(k)];
    deltaT = (Time(k+1) - Time(k)) * 3600*24; % convert to seconds

    %Measurement Update
    K_k = PkApriori*Hk' * (Hk*PkApriori*Hk' + Rk)^-1;
    Xk(k,:) = XkApriori + K_k*(zk - (Hk*XkApriori));
    Pk = (eye(4,4) - K_k*Hk)*PkApriori;

    %Time update; recalculate phi and Q based on delta T
    Phi = [[1 deltaT 0 0]
           [0 1 0 0]
           [0 0 1 deltaT]
           [0 0 0 1]];

    Q = [[(W/3)*deltaT^3 (W/2)*deltaT^2 0 0]
        [(W/2)*deltaT^2 W*deltaT 0 0]
        [0 0 (W/3)*deltaT^3 (W/2)*deltaT^2]
        [0 0 (W/2)*deltaT^2 W*deltaT]];

    XkApriori = Phi*Xk(k,:)';
    PkApriori = Phi*Pk*Phi' + Q;


end
```

The filter is implemented in a for loop that goes from the first data point to the second-to-last data point. The last data point is ignored because there is not enough data to calculate a delta T value. Before the loop starts, we must initialize the first value of Xk by using our best educated guess. The initial positions are set to the first data point, and the initial velocities are set to zero. The guesses for velocities can later be changed to improve filter performance. During each step of the for-loop, the code first obtains the data point and delta T value. Then, it computes the three measurements update equations to get current estimates for Xk. Lastly, it performs the time update by computing the Phi and Q matrices and using them to get the next a-priori estimate of Xk. Once the loop has finished going through all the data, the Xk variable will have the estimates for all four state variables, which are plotted in the results section below. The full code is included in the appendix for more information.

Results





Discussion

Tuning parameters:

$\boldsymbol{R_K}$

The measurement noise covariance matrix was tuned while keeping W unity, $\hat{x}_0^- = [p_x(0) \quad 0 \quad p_y(0) \quad 0]'$, and

$$\hat{P}_0^- = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Table 1. Measurement Noise Covariance Matrix Observations

Value	Observations
$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	The position estimate matches exactly the observation data and there is no longer any velocity estimates. This occurs because the measurement is not completely "trusted" by the model since there is no uncertainty of the measurement indicated by a 0 value in R_K.
$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	The position estimate matches very closely the observation data and there are now estimates for velocity in both the x and y direction. The velocity data seems relatively noisy.
$\begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$	The position estimate is becoming a smoother function of the observation data. The velocities seem to have less variance.
$\begin{bmatrix} >10 & 0 \\ 0 & >10 \end{bmatrix}$	As the first and fourth elements increase in order of magnitude the smoother the position and velocity signals become. Over the 8 minutes of position data provided, the course of the submarine should be relatively steady and constant velocity. The velocity estimates become more constant after a certain point where the model must catch up after having a bad initial estimate of velocity.

$\boldsymbol{W}$

The process noise was tuned while keeping $R_K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\hat{x}_0^- = [p_x(0) \quad 0 \quad p_y(0) \quad 0]'$, and

$$\hat{P}_0^- = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Table 2. Process Noise Observations

Value	Observations
0	Having no process noise in the system gives a completely linear position estimation function and 5%-10% of the iterations produces a constant x,y velocity.
1	There is no longer a linear estimation of position and now looks similar to the measured data. Additionally, there are noisy velocity estimations in both x and y directions.
>1	The estimations for position become closer matching to the measured position data. The velocity estimates have increases in variance (noisier).

$\hat{x}_0^-$

The initial state estimation *a priori* was tuned while keeping $R_K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, W unity, and

$$\hat{P}_0^- = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Table 3. A Priori Initial State Estimation Observations

Value	Observation
$[0 \quad 0 \quad 0 \quad 0]'$	Low initial position (compared to the actual initial position) for the system causes overshoot after a few iterations but then corrects rather quickly to resemble the measured data. Causes large overshoot of velocity initial estimations and then becomes a constant 0 in both directions.
$[p_x(0) \quad 0 \quad p_y(0) \quad 0]'$	No undershoot or overshoot, the position estimates are very similar to the measured data. The velocity is noisy with an average close to 0 in both directions.
$[1000000 \quad 0 \quad 1000000 \quad 0]'$	High initial position (compared to the actual initial position) performs similarly to the 0 initial position case, but has a large undershoot instead of overshoot
$\left[p_x(0) \quad \frac{\lvert p_x(1) - p_x(0)\rvert}{\Delta T} \quad p_y(0) \quad \frac{\lvert p_y(1) - p_y(0)\rvert}{\Delta T}\right]'$	*Ideal, but not feasible for real time

$\hat{P}_0^-$

The initial error covariance matrix *a priori* was tuned while keeping $R_K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\hat{x}_0^- = [p_x(0) \quad 0 \quad p_y(0) \quad 0]'$, and W unity.

Table 4. A Priori Initial Error Covariance Matrix Observations

Value	**Observations**
$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$	Nearly identical position estimations to measured data. Noisy velocity estimations.
$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$	Similar observations to above
$\begin{bmatrix} 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 \end{bmatrix}$	Similar observations to above

$\begin{bmatrix} Z^+ & 0 & 0 & 0 \\ 0 & Z^+ & 0 & 0 \\ 0 & 0 & Z^+ & 0 \\ 0 & 0 & 0 & Z^+ \end{bmatrix}$	Similar observations to above

The measurement noise covariance matrix has influence on the smoothness of the overall estimations. The larger the uncertainty about the measurement noise the more “noise” gets filtered out of the signal. The smaller the uncertainty about the measurement noise the more the system “trusts” the measurement and corrects less.

In absence of process noise the Kalman filter relies on system design for estimations, namely, the PV model used in this case. As process noise increases, there is greater noise that passes through to the estimations of velocity and the estimations of the positions are closer to the measured position data.

The initial *a priori* state estimation will set up how much the filter has to correct the signal. This parameter is also closely dependent on the *a priori* error covariance matrix because of the calculation of the Kalman gain and then the present estimation of the state vector. It is ideal to not have a lot of overshoot or undershoot in the estimation. Additionally, having initial information for the velocity would help the undershoot/overshoot of the velocity estimations but was unavailable for this case.

The *a priori* error covariance matrix did not have a lot of influence on the estimations. This is unusual because the error covariance matrix drives the Kalman filter to correct in certain ways.

In conclusion, the selection of parameters should be chosen based on getting the estimations as close to the actual data as possible. In this case, there was no provided actual data, but the filter was optimized for smoothness and linear estimations for position and constant estimations for velocity. Therefore, the following parameters were used: $R_K = \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}$, W = 1, $\hat{x}_0^- = [p_x(0) \quad 0 \quad p_y(0) \quad 0]'$, and $\hat{P}_0^- = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

3D Kalman Filters

1. Set Up of the 3D Kalman Filter:

The setup for the 3D Kalman filter is very similar to the 2D case. The only difference is we are adding a z position and velocity component to our state vector.

$$\bar{x} = \begin{bmatrix} p_x(t) \\ v_x(t) \\ p_y(t) \\ v_y(t) \\ p_z(t) \\ v_z(t) \end{bmatrix}$$

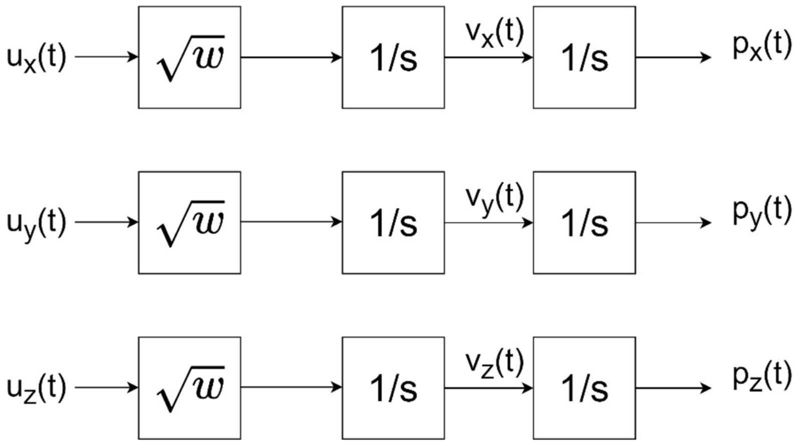


Figure 2 – 3D Kalman Filter System Model

Following the same steps listed in the 2D Kalman filter derivation the discretized state transition matrix for the 3D system can be shown to be:

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

Similiarly since our measurment vector (z_k) contains only positional measurments:

$$z_k = \boldsymbol{H}_k x_k + v_k$$

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

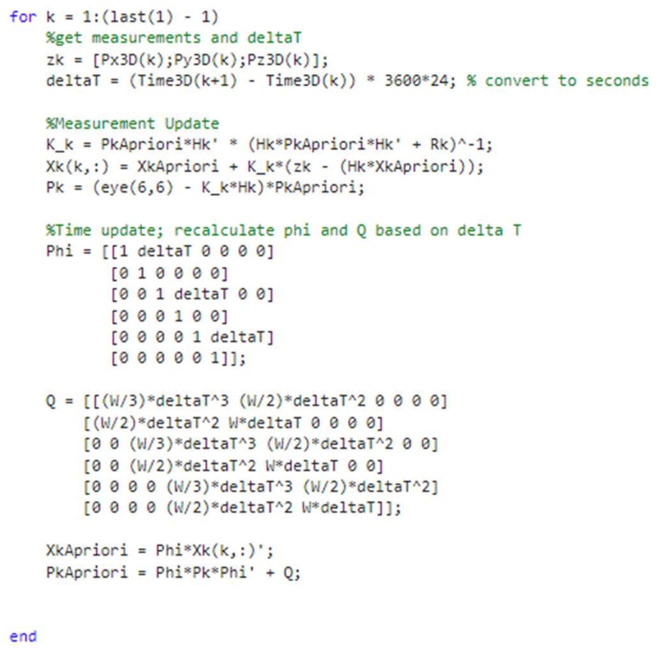
Taking the covariance of the proccess and measurment noise results in:

$$\boldsymbol{Q}_k = \begin{bmatrix} \frac{w}{3}\Delta t^3 & \frac{w}{2}\Delta t^2 & 0 & 0 & 0 & 0 \\ \frac{w}{2}\Delta t^2 & w\Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{w}{3}\Delta t^3 & \frac{w}{2}\Delta t^2 & 0 & 0 \\ 0 & 0 & \frac{w}{2}\Delta t^2 & w\Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{w}{3}\Delta t^3 & \frac{w}{2}\Delta t^2 \\ 0 & 0 & 0 & 0 & \frac{w}{2}\Delta t^2 & w\Delta t \end{bmatrix}$$

$$R_k = \begin{bmatrix} q\Delta t & 0 & 0 \\ 0 & q\Delta t & 0 \\ 0 & 0 & q\Delta t \end{bmatrix}$$

2. Implementation of the 3D Kalman filter:

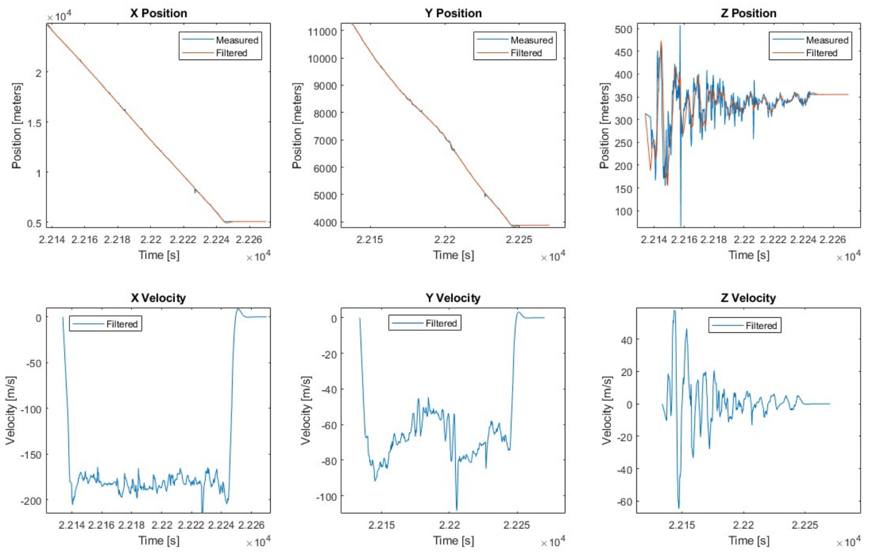
The implementation of the 3D Kalman filter is very similar to the 2D case. The only differences between the two are the dimensions of all the matrices and the Hk, Phi, and Q matrices. The code for the 3D implementation is shown below:

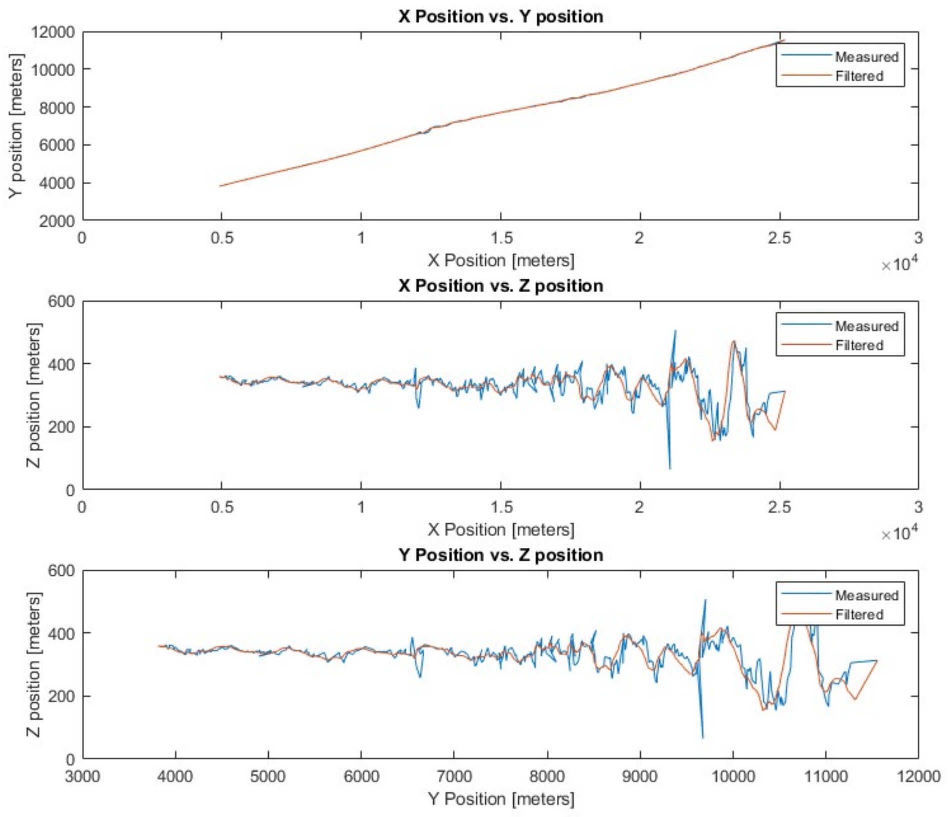


```
for k = 1:(last(1) - 1)
    %get measurements and deltaT
    zk = [Px3D(k);Py3D(k);Pz3D(k)];
    deltaT = (Time3D(k+1) - Time3D(k)) * 3600*24; % convert to seconds

    %Measurement Update
    K_k = PkApriori*Hk' * (Hk*PkApriori*Hk' + Rk)^-1;
    Xk(k,:) = XkApriori + K_k*(zk - (Hk*XkApriori));
    Pk = (eye(6,6) - K_k*Hk)*PkApriori;

    %Time update; recalculate phi and Q based on delta T
    Phi = [[1 deltaT 0 0 0 0]
           [0 1 0 0 0 0]
           [0 0 1 deltaT 0 0]
           [0 0 0 1 0 0]
           [0 0 0 0 1 deltaT]
           [0 0 0 0 0 1]];

    Q = [[(W/3)*deltaT^3 (W/2)*deltaT^2 0 0 0 0]
        [(W/2)*deltaT^2 W*deltaT 0 0 0 0]
        [0 0 (W/3)*deltaT^3 (W/2)*deltaT^2 0 0]
        [0 0 (W/2)*deltaT^2 W*deltaT 0 0]
        [0 0 0 0 (W/3)*deltaT^3 (W/2)*deltaT^2]
        [0 0 0 0 (W/2)*deltaT^2 W*deltaT]];

    XkApriori = Phi*Xk(k,:)';
    PkApriori = Phi*Pk*Phi' + Q;


end
```

The code works in the same way as the 2D case, however, there are a few differences to note. First, we now have to obtain three measurements for each iteration of the loop: Px, Py, Pz. The dimensions of all the variables have been increased to match the addition of two new state variables, notably, the identity matrix used in the computation of Pk is now 6X6 instead of 4X4. Lastly, the Phi and Q matrices have been changed to the matrices derived in the section above; they are now also 6X6 matrices. The full code is included in the appendix for more information.

Results


X Position
Y Position
Z Position
Measured
Filtered
Position [meters]
Time [s]
×10^4
X Velocity
Y Velocity
Z Velocity
Velocity [m/s]
Filtered


X Position vs. Y position
Y position [meters]
X Position [meters]
×10^4
X Position vs. Z position
Z position [meters]
Y Position vs. Z position
Y Position [meters]
Measured
Filtered

Discussion

Tuning parameters:

$\boldsymbol{R_K}$

The measurement noise covariance matrix was tuned while keeping W unity, $\hat{x}_0^- = [p_x(0) \quad 0 \quad p_y(0) \quad 0 \quad p_z(0) \quad 0]'$, and

$$\hat{P}_0^- = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Table 5. Measurement Noise Covariance Matrix Observations

Value	Observations
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	The position estimate matches exactly the observation data (for x and y position data the estimates and measured data are linear which is desirable). The velocity estimates in all directions are noisy.
$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	The x and y position estimates match very closely to the measured data; the position estimates for the z-direction are tighter to a constant value. The velocity estimates are less noisy than previous case.
$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}$	X and y position and velocity estimate (in all directions) observations are similar to previous case. In the z-direction, position estimates are getting noisier in the beginning of the iterations but gets smoother than the measured data as time goes on.
$\begin{bmatrix} >10 & 0 & 0 \\ 0 & >10 & 0 \\ 0 & 0 & >10 \end{bmatrix}$	As the elements along the diagonal get bigger, (1) X and y position estimates are rounding sharp changes in direction and ignores small noise artifacts (2) Z position estimates are becoming less noisy, but still have a damping shape (3) Velocity estimates are becoming less noisy. X velocity becomes more constant after correction from initial velocity. Y and z velocities are relatively non-constant in shape (4) >100000 shows more linear positions and more constant velocities

$\boldsymbol{W}$

The process noise was tuned while keeping $R_K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $\hat{x}_0^- = [p_x(0) \quad 0 \quad p_y(0) \quad 0 \quad p_z(0) \quad 0]'$, and

$$\hat{P}_0^- = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Similar observations can be found in *Table 2*.

$\boldsymbol{\hat{x}_0^-}$

The initial state estimation *a priori* was tuned while keeping $R_K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, W unity, and

$$\hat{P}_0^- = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Table 6. A Priori Initial State Estimation Observations

Value	Observation
$\hat{x}_0^- = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]'$	Low initial position (compared to the actual initial position) for the system causes overshoot after a few iterations but then corrects rather quickly to resemble the measured data. Causes large overshoot of velocity initial estimations and then becomes a constant 0 in x, y directions. Z velocity estimates show a signal with high variance and damping as time goes on.
$[p_x(0) \quad 0 \quad p_y(0) \quad 0 \quad p_z(0) \quad 0]'$	No undershoot or overshoot, the position estimates are very similar to the measured data. The velocity is noisy with an average close to -175, -50, and 0 in the x-, y-, and z-directions, respectively. The z-direction velocity estimate also has a damping signal.
$[100000 \quad 0 \quad 100000 \quad 0 \quad 100000 \quad 0]'$	High initial position (compared to the actual initial position) performs similarly to the 0 initial position case but has a large undershoot instead of overshoot. All velocity estimates show a large negative spike and then approximately zero velocity.
$\begin{bmatrix} p_x(0) \\ \frac{\lvert p_x(1) - p_x(0) \rvert}{\Delta T} \\ p_y(0) \\ \frac{\lvert p_y(1) - p_y(0) \rvert}{\Delta T} \\ p_z(0) \\ \frac{\lvert p_z(1) - p_z(0) \rvert}{\Delta T} \end{bmatrix}$	*Ideal, but not feasible for real time

$\boldsymbol{\hat{P}_0^-}$

The initial error covariance matrix *a priori* was tuned while keeping $R_K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $\hat{x}_0^- =$ $[p_x(0) \quad 0 \quad p_y(0) \quad 0 \quad p_z(0) \quad 0]'$, and W unity.

Table 7. A Priori Initial Error Covariance Matrix Observations

Value	**Observations**
$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	Nearly identical position estimations to measured data. Noisy velocity estimations. Z-direction position estimates are slightly less noisy than measured data.
$\begin{bmatrix} Z^+ & 0 & 0 & 0 & 0 & 0 \\ 0 & Z^+ & 0 & 0 & 0 & 0 \\ 0 & 0 & Z^+ & 0 & 0 & 0 \\ 0 & 0 & 0 & Z^+ & 0 & 0 \\ 0 & 0 & 0 & 0 & Z^+ & 0 \\ 0 & 0 & 0 & 0 & 0 & Z^+ \end{bmatrix}$	Similar observations to above, not much change as the diagonal elements increase even at much larger magnitudes.

The measurement noise covariance matrix has influence on the smoothness of the overall estimations. Many of the same conclusions can be drawn as seen in the 2D case, however, for the 3D case there needed to be much larger values at the diagonal elements to achieve smoother data. The data for the 3D case also seemed to be much noisier, especially in the z-direction.

In absence of process noise the Kalman filter relies on system design for estimations, namely, the PV model used in this case. As process noise increases, there is greater noise that passes through to the estimations of velocity and the estimations of the positions are closer to the measured position data.

The same conclusions can be drawn for the initial *a priori* state estimation matrix for the 3D case as in the 2D case. An additional observation for this parameter was that the overshoot and undershoot was greater in amplitude than in the 2D case.

The *a priori* error covariance matrix did not have a lot of influence on the estimations. This is unusual because the error covariance matrix drives the Kalman filter to correct in certain ways.

In conclusion, the selection of parameters should be chosen based on getting the estimations as close to the actual data as possible. In this case, there was no provided actual data, but the filter was optimized for smoothness and linear estimations for position and constant estimations for velocity. Therefore, the following parameters were used: $R_K = \begin{bmatrix} 100000 & 0 & 0 \\ 0 & 100000 & 0 \\ 0 & 0 & 100000 \end{bmatrix}$, W = 1, $\hat{x}_0^- =$ $[p_x(0) \quad 0 \quad p_y(0) \quad 0 \quad p_z(0)\ 0]'$, and

$$\hat{P}_0^- = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Appendix

Note: There is a MATLAB data file for each script that needs to be included in the workspace before running. These files are DATA2D.mat and DATA3D.mat and they contain the data from the txt files given.

2D Code

```
%% 2D Kalman filter
%This code needs the Px, Py, and Time data already in the workspace
clc
close all

last = size(Px);
W = 0.001;
deltaT = 1;
Hk = [1 0 0 0;0 0 1 0];

XkApriori = [Px(1) 0  Py(1) 0]';
PkApriori = [[1 0 0 0]
             [0 1 0 0]
             [0 0 1 0]
             [0 0 0 1]];

Rk = [1 0 ; 0 1];

Xk = zeros(4, last(1) - 1)';

for k = 1:(last(1) - 1)
    %get measurements and deltaT
    zk = [Px(k);Py(k)];
    deltaT = (Time(k+1) - Time(k)) * 3600*24; % convert to seconds

    %Measurement Update
    K_k = PkApriori*Hk' * (Hk*PkApriori*Hk' + Rk)^-1;
    Xk(k,:) = XkApriori + K_k*(zk - (Hk*XkApriori));
    Pk = (eye(4,4) - K_k*Hk)*PkApriori;

    %Time update; recalculate phi and Q based on delta T
    Phi = [[1 deltaT 0 0]
           [0 1 0 0]
           [0 0 1 deltaT]
           [0 0 0 1]];

    Q = [[(W/3)*deltaT^3 (W/2)*deltaT^2 0 0]
        [(W/2)*deltaT^2 W*deltaT 0 0]
        [0 0 (W/3)*deltaT^3 (W/2)*deltaT^2]
        [0 0 (W/2)*deltaT^2 W*deltaT]];

    XkApriori = Phi*Xk(k,:)';
    PkApriori = Phi*Pk*Phi' + Q;


end

%Do not plot the last sample data; it is ignored due to the lack of a
```

```
%deltaT
time = Time(1:end-1);
px = Px(1:end-1);
py = Py(1:end-1);

figure;
subplot(2,2,1);
plot(time,px, time, Xk(:,1));
ylim([87650 87750]);
title("Position X");
legend("Measured", "Filtered");

subplot(2,2,2);
plot(time,py, time,Xk(:,3));
title("Position Y");
legend("Measured", "Filtered");

subplot(2,2,3);
plot(time,Xk(:,2));
title("Velocity x");
legend("Filtered");

subplot(2,2,4);
plot(time,Xk(:,4));
title("Velocity y");
legend("Filtered");
```

3D code

```
%% 3D Kalman Filter

%This code needs the Px3D, Py3D, Pz3D and Time data already in the workspace
clc
close all

last = size(Px3D);
W = 1;
deltaT = 1;
Hk = [[1 0 0 0 0 0]
      [0 0 1 0 0 0]
      [0 0 0 0 1 0]];

XkApriori = [Px3D(1) 0  Py3D(1) 0 Pz3D(1) 0]';
PkApriori = [[10 10 10 10 10 10]
             [10 10 10 10 10 10]
             [10 10 10 10 10 10]
             [10 10 10 10 10 10]
             [10 10 10 10 10 10]
             [10 10 10 10 10 10]];

Rk = [[100 0 0]
      [0 100 0]
      [0 0 100]];
```

```
Xk = zeros(6, last(1) - 1)';

for k = 1:(last(1) - 1)
    %get measurements and deltaT
    zk = [Px3D(k);Py3D(k);Pz3D(k)];
    deltaT = (Time3D(k+1) - Time3D(k)) * 3600*24; % convert to seconds

    %Measurement Update
    K_k = PkApriori*Hk' * (Hk*PkApriori*Hk' + Rk)^-1;
    Xk(k,:) = XkApriori + K_k*(zk - (Hk*XkApriori));
    Pk = (eye(6,6) - K_k*Hk)*PkApriori;

    %Time update; recalculate phi and Q based on delta T
    Phi = [[1 deltaT 0 0 0 0]
           [0 1 0 0 0 0]
           [0 0 1 deltaT 0 0]
           [0 0 0 1 0 0]
           [0 0 0 0 1 deltaT]
           [0 0 0 0 0 1]];

    Q = [[(W/3)*deltaT^3 (W/2)*deltaT^2 0 0 0 0]
        [(W/2)*deltaT^2 W*deltaT 0 0 0 0]
        [0 0 (W/3)*deltaT^3 (W/2)*deltaT^2 0 0]
        [0 0 (W/2)*deltaT^2 W*deltaT 0 0]
        [0 0 0 0 (W/3)*deltaT^3 (W/2)*deltaT^2]
        [0 0 0 0 (W/2)*deltaT^2 W*deltaT]];

    XkApriori = Phi*Xk(k,:)';
    PkApriori = Phi*Pk*Phi' + Q;


end

%Do not plot the last sample data; it is ignored due to the lack of a
%deltaT
time3D = Time3D(1:end-1);
px3D = Px3D(1:end-1);
py3D = Py3D(1:end-1);
pz3D = Pz3D(1:end-1);

figure;
subplot(2,3,1);
plot(time3D,px3D, time3D, Xk(:,1));
% ylim([87650 87750]);
title("Position X");
legend("Measured", "Filtered");

subplot(2,3,2);
plot(time3D,py3D, time3D,Xk(:,3));
title("Position Y");
legend("Measured", "Filtered");

subplot(2,3,3);
plot(time3D,pz3D, time3D, Xk(:,5));
```

```
% ylim([87650 87750]);
title("Position Z");
legend("Measured", "Filtered");


subplot(2,3,4);
plot(time3D,Xk(:,2));
title("Velocity x");
legend("Filtered");

subplot(2,3,5);
plot(time3D,Xk(:,4));
title("Velocity y");
legend("Filtered");

subplot(2,3,6);
plot(time3D,Xk(:,6));
title("Velocity z");
legend("Filtered");
```