Effects of Measurement Availability on Extended Kalman Filtering

Ramzi I. Kattan *
School of Aeronautics and Astronautics, 701 W Stadium Ave, West Lafayette, IN.

Recent years have seen the increase in tracking systems that utilize camera and radar sensor to trace the 3 dimensional trajectory of a golf ball, with programs such as TrackMan and TopTracer. These systems all employ specific tracking algorithms to predict flight trajectory when measurements are unavailable. This study uses the Extended Kalman Filter to trace the flight using 5 different cases of measurement availability. All cases are able to predict the trajectory of the ball with relative accuracy. However, cases with measurements received at longer intervals of time trace a trajectory with jumps in space, an unexpected behavior. Cases that receive few measurements at the start of the flight produce no jumps, but poorly predict final states. Each case has is best suited for different types of golfing experiences. The results of the study require further investigation using real trajectory data from driving ranges, as well as a more updated model that makes fewer assumptions on the modeled trajectory. Future studies will include multi-object multi-sensor tracking, similar to more advanced and accurate systems employed today.

I. Introduction

The motivation to apply Extended Kalman Filtering to the sport of golf arose from the mainstream implementation of multi-sensor multi-object golf ball tracking systems. The PGA Tour announced in 2022 that they will implementing TrackMan technology on their courses in order to "help enrich the fan experience" [1]. TrackMan systems use highly advanced Doppler Radar technology to track club and ball more accurately than any of their competitors [2]. Although the most accurate systems, TrackMan systems are fairly expensive and provide more data and tracking capabilities than the average player requires. It is for this reason that most driving ranges, tailored to causal players, use TopTracer software instead. Toptracer has "traced more balls in more bays at more driving ranges in more countries than any other range technology on the planet" [3]. TopTracer systems also track trajectory of a ball, using simpler solutions that are relatively less accurate and provide less data.

*Aeronautical and Astronautical Senior, School of Aeronautics and Astronautics, 701 W Stadium Ave, West Lafayette, IN.

A. How Does TopTracer Work

Inventor Daniel Forsgren [4] patented TopTracer in 2018 to provide players at institutions such as TopGolf a more immersive and informed experience. TopTracer uses a camera and radar system to trace the trajectory of a golf ball using the following equation:

$$R_n = R_{n-1} + S_n * dt \tag{1}$$

$$Z_n = R_n * cos(\theta_n) \tag{2}$$

$$X_n = Z_n * dx_n / f \tag{3}$$

$$Y_n = Z_n * dy_n / f \tag{4}$$

where:

R_n	=	Range at observation n (in units of length)
S_n	=	Radial Speed at observation n (in units of length / time)
dt	=	Time between consecutive observations (in unit of time)
Z_n	=	Z Cartesian coordinate of ball at observation n (in units of length)
X_n	=	X Cartesian coordinate of ball at observation n (in units of length)
Y_n	=	Y Cartesian coordinate of ball at observation n (in units of length)
θ_n	=	Angle between camera sensor center and location of ball impression on camera at observation n (in units of angle)
dx_n	=	Horizontal offset (in pixels) between the ball location on sensor and the sensor center at observation n
dy_n	=	Vertical offset (in pixels) between the ball location on sensor and the sensor center at observation n
f	=	Focal length of camera optics (in pixels)

These equations are only valid if the radar and camera are assumed to at the same location. When this data is collected, a real-time filtering technique is applied to simulate the entire trajectory of a golf ball being hit. Different constraints can be applied to the system such that it only tracks objects hit with certain initial speeds (resulting in some balls not being tracked if their initial speeds are lower than the software can intake). The filtering technique for this process is not discussed in the patent, but an Extended Kalman Filter may be employed for such a problem.

B. Objective and Scope of this Study

The objective of this study is to employ a similar technique to that of TopTracer using an Extended Kalman Filter, with slightly different measurement techniques and predefined flight dynamics using a trajectory model found from literature. This study aims to look at how the Extended Kalman Filter performs in various limited measurement availability cases.

C. Structure of Report

This report will first formulate the problem by identifying the problem statement and significance of the study. The methodology section of this report will describe the approach used to model dynamics, derivation and initialization of the filter, and how performance is assessed. Results and findings will then be shared and discussed with limitations of the study. Implications of the findings will be summarized with a plan to continue the study, if possible. All code used for the study can be found in the appendix along with references.

II. Problem Formulation

This study will utilize the Extended Kalman Filter (EKF) to track the nonlinear dynamics of a golf ball after it has been struck to the first instant it touches the ground. This study assess the performance of the EKF under various (5) cases of measurement availability provided by the senors.

A. Choice of Extended Kalman Filter

The decision to employ an EKF was due to the nonlinear nature of the the system. The dynamics of the trajectory do not follow linear models, and neither do they measurement models. EKFs linearize the dynamic model at every instant in time, and then the principles of Kalman filtering are applied to the instantaneous linear models. This linearization is analytic in nature and allows for uncertainties to be projected and maintain their Gaussian shape (an important assumption). An Unscented Kalman Filter (UKF) can also be employed. UKF's are better suited for systems with high nonlinearity because they do not try to find the statistics of the system by approximating nonlinear functions using Tayler Series Expansion, the method used by EKFs. UKFs instead directly approximate the mean and covariance of the target distribution. However, because of this, UKFs are difficult to employ and more computationally demanding. An attempt at a UKF was made and will be discussed.

B. Relevance and Significance

The relevance of this study can be attributed to the increasing demand in trajectory tracking in the sport of golf. With more facilities wanting to provide similar experiences as TopGolf and as seen on TV, there is a need for more redundant systems in the market that provide this capability for a cheaper cost. One way costs can be cheaper, is by decreasing the computational rigor of such systems. If it can be found that relatively accurate systems can be produced by applying Extended Kalman Filters with minimal measurement availability, then more facilities may be able to equip themselves with this experience.

III. Methodology

In order to demonstrate how this problem will be solved, assumptions must first be made to apply the EKF. The following are a list of assumptions made:

1) Golf flight trajectory can be accurately modeled using differential equations.
2) Process and measurement noise follow Gaussian Distribution.
3) The system can be accurately linearized using the first-order Taylor Series.
4) There are relatively small nonlinear effects.

These assumptions are all crucial to apply an EKF. Next, the model will be presented and described.

A. Golf Ball Trajectory Model

The following model was derived by Brett Burglund and Ryan Street ([5]), the following model is based on their derivation:

$$\ddot{x} = -\frac{D}{m}\dot{x}^2 + \frac{S}{m}(\omega_j \dot{z} - \omega_k \dot{y}) \tag{5}$$

$$\ddot{y} = -g - \frac{D}{m}\dot{y}^2 + \frac{S}{m}(\omega_k \dot{x} - \omega_i \dot{z}) \tag{6}$$

$$\ddot{z} = -\frac{D}{m}\dot{z}^2 + \frac{S}{m}(\omega_i \dot{y} - \omega_j \dot{x}) \tag{7}$$

With

$$D = \frac{1}{2} C_d \rho \tag{8}$$

Where

$\ddot{x}$	=	Acceleration in the x-direction (ft/sec^2)
$\ddot{y}$	=	Acceleration in the y-direction (ft/sec^2)
$\ddot{z}$	=	Acceleration in the z-direction (ft/sec^2)
$\dot{x}$	=	Velocity in the x-direction (ft/sec)
$\dot{y}$	=	Velocity in the y-direction (ft/sec)
$\dot{z}$	=	Velocity in the z-direction (ft/sec)
m	=	Mass of the golf ball (lbs)
$\dot{z}$	=	Velocity in the z-direction (ft/sec)
ω	=	Spin rate (angular velocity) about i,j,k axis (rps)
S	=	Magnus Coefficient
D	=	Drag Multiplier (lbs/in)
ρ	=	Density of air

C_d = Drag Coefficient

This model includes a few crucial assumptions to make modeling the dynamics of a golf ball easier:

1) Spin rate (angular velocity) is constant.
2) Magnus Coefficient is known.
3) No changes in air conditions due to altitude or humidity.
4) No changes in pressure.

Although these assumptions are not realistic, their effects will be included in the determination of process noise. The EKF can now be defined.

B. The Extended Kalman Filter

The notion and method of the EKF used in this study was provided by Assistant Professor Keith LeGrand's AAE590ET lecture notes[6][7][8][9]. A more detailed explanation of the theory and derivation of the filter can be found in the references. The Continuous-Discrete Extended Kalman Filter is described next.

1. System Dynamics

The system dynamics are described using equations (5), (6), and (7) in the form of:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), t) + G(t)\boldsymbol{w}(t) \tag{9}$$

If $x_1 = x$ $x_2 = \dot{x}$ $x_3 = y$ $x_4 = \dot{y}$ $x_5 = z$ $x_6 = \dot{z}$. Then:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \\ \dot{x_5} \\ \dot{x_6} \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{D}{m}x_2^2 + \frac{S}{m}(\omega_j x_6 - \omega_k x_4) \\ x_4 \\ -g - \frac{D}{m}x_4^2 + \frac{S}{m}(\omega_k x_2 - \omega_i x_6) \\ x_6 \\ -\frac{D}{m}x_6^2 + \frac{S}{m}(\omega_i x_4 - \omega_j x_2) \end{bmatrix} + [G][w] \tag{10}$$

Where:

$\boldsymbol{f}(\boldsymbol{x}(t), t)$ = Nonlinear Dynamics System
G = Is the Process Noise Mapping Matrix
w = Process Noise

For this study, process noise will only be applied to acceleration. This was done in order to maximize tuning ability, since process noise in acceleration trickles down into velocity and acceleration. Therefore, the process noise mapping matrix is defined as:

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

For a unique model such as that in equation 10, no process noise could be found in literature. However, various factors such as wind speed and direction, air pressure, humidity, temperature, surface type, ball type, and spin evolution all have effects on the trajectory of the ball. Due to the exact process noise being unknown, the order of magnitude will be reasoned. The most dominant factor affecting the flight of the golf ball is wind. It's assumed that this wind occurs randomly in the x and z directions of the golf ball, therefore their distribution will have a standard deviation one order of magnitude greater than that in the y direction. The rest of the factors are relatively minuscule over the course of the flight. One method to finding process noise is to plot multiple trajectories with varying process noise. Due to the high level of uncertainty in determining process noise, its fair to pick the largest process noise that still displays the expected flight of a golf ball, since it follows a known trajectory.

Assume $\omega_x = \omega_z \sim (0, (10^i)^2)$ and $\omega_y \sim (0, (10^{i-1})^2)$. Plot for $i = [-5, 3]$.

From the figure 1, the effects of process noise with i = 2, 3 greatly differs from the other trajectories, so they will be removed to assess the process noise effects.

From from figure 2, i = 0 produces a trajectory one may expect from a golf ball. Therefore, i = 0 will be used for the process noise definition. With this, the process noise will be $\omega_x = \omega_z \sim (0, (1)^2)$ and $\omega_y \sim (0, (10^{-1})^2)$. The process noise vector is:

$$\omega = \begin{bmatrix} N(0, 1^2) \\ N(0, (10^{-1}))^2 \\ N(0, 1^2) \end{bmatrix} \tag{11}$$

Therefore, our systems dynamics can be described in full:

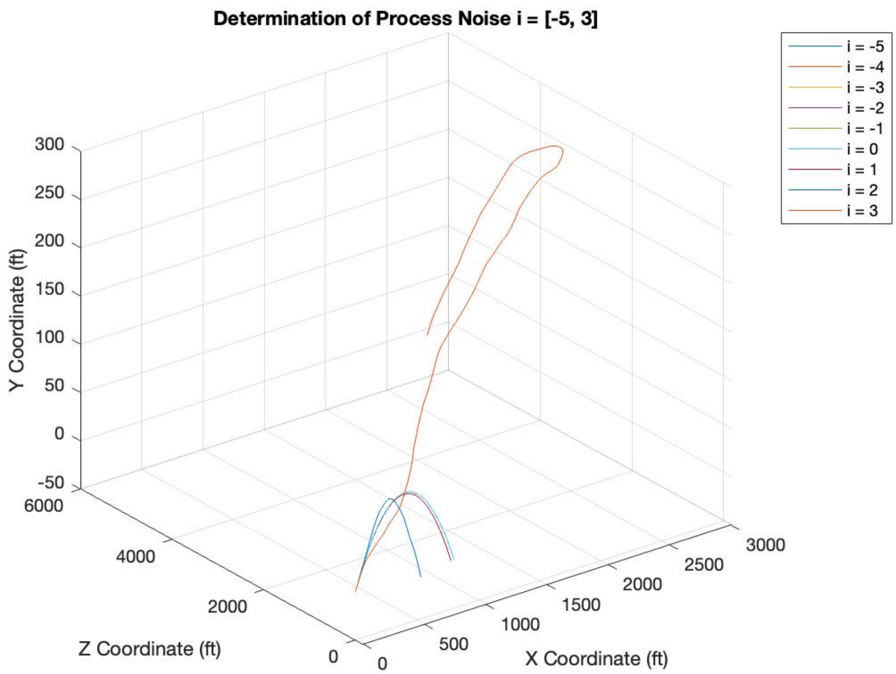

Fig. 1 Process Noise Determination for i = [-5, 3].

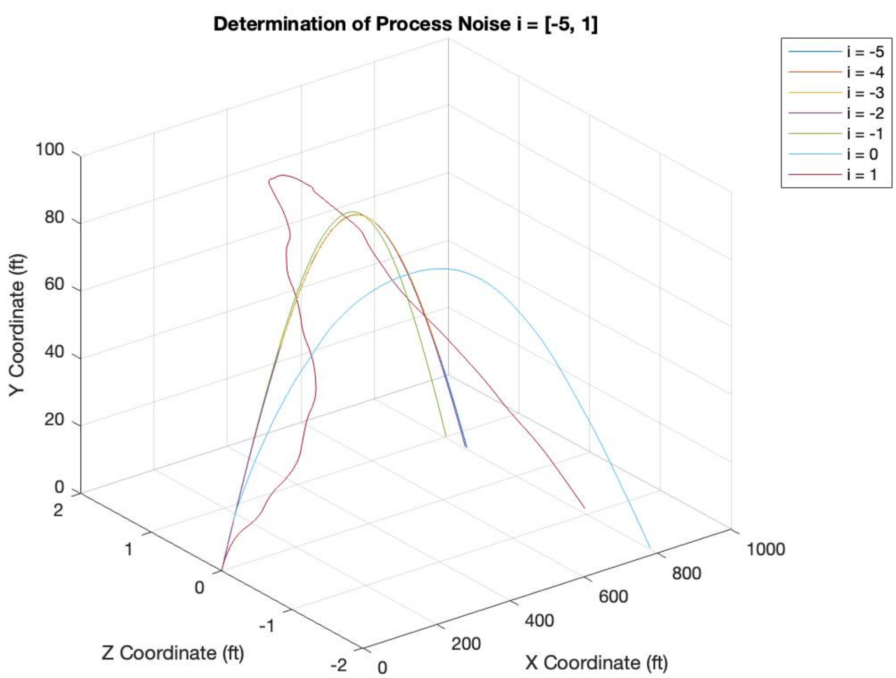

Fig. 2 Process Noise Determination for i = [-5, 1].

$$
\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \\ \dot{x_5} \\ \dot{x_6} \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{D}{m}x_2^2 + \frac{S}{m}(\omega_j x_6 - \omega_k x_4) \\ x_4 \\ -g - \frac{D}{m}x_4^2 + \frac{S}{m}(\omega_k x_2 - \omega_i x_6) \\ x_6 \\ -\frac{D}{m}x_6^2 + \frac{S}{m}(\omega_i x_4 - \omega_j x_2) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} N(0, 1^2) \\ N(0, (10^{-1}))^2 \\ N(0, 1^2) \end{bmatrix} \tag{12}
$$

For this system, a golf player strikes the ball with an initial true state of:

$$
\boldsymbol{x_0} = \begin{bmatrix} 0 \\ 180 \\ 0 \\ 90 \\ 0 \\ 10 \end{bmatrix} \tag{13}
$$

The rest of the flight parameters are defined in the appendix. With the system dynamics modeled, the measurement model must be defined as well.

2. Measurement Model

The measurement model describes how measurements are calculated from the current state. For the EKF, the measurement model is in the form:

$$
z_k = \boldsymbol{h(x_k)} + \boldsymbol{v_k} \tag{14}
$$

Where

$\boldsymbol{f}(\boldsymbol{x}(t), t)$	=	Nonlinear Dynamics System
$\boldsymbol{z_k}$	=	Measurement received at observation k
$\boldsymbol{x_k}$	=	Current true state
$\boldsymbol{h(x_k)}$	=	Measurement function
$\boldsymbol{v_k}$	=	Measurement noise

For this problem, it is assumed that measurements are taken using a dual camera-radar sensor, such as that used by TopTracer. It is assumed that the sensors can observe 4 states, range, radial speed [10], offset angle, and inclination angle. These 4 states are defined by the true state $\boldsymbol{x_k}$ as:

$$\begin{bmatrix} R \\ S \\ \theta \\ \phi \end{bmatrix} = \begin{bmatrix} \sqrt{x_1^2 + x_3^2 + x_5^2} \\ \frac{x_1 x_2 + x_3 x_4 + x_5 x_6}{\sqrt{x_1^2 + x_3^2 + x_5^2}} \\ \arctan \frac{x_5}{x_1} \\ \arccos \frac{x_3}{\sqrt{x_1^2 + x_3^2 + x_5^2}} \end{bmatrix} \tag{15}$$

R = Range

S = Radial speed.

θ = Azimuth angle made between x-axis and velocity z-component .

ϕ = Polar angle made between y-axis and velocity vector.

The measurement model requires process noise. According to BlackBoxGolf, the most accurate sensors are able to measure range within 1 yard at a distance of 100 yards ([11]). Therefore, $v_R \sim N(0, (1)^2)$. For the remaining noise, it is assumed that the standard deviation is on the order of 5% of maximum values for the same initial conditions as in figure 1 and 2. The measurement noise is therefore:

$$\boldsymbol{v} = \begin{bmatrix} N(0, (1)^2) \\ N(0, (10)^2) \\ N(0, (10^{-3})^2) \\ N(0, (10^{-1})^2) \end{bmatrix} \tag{16}$$

Therefore, the measurement model is:

$$\begin{bmatrix} R \\ S \\ \theta \\ \phi \end{bmatrix} = \begin{bmatrix} \sqrt{x_1^2 + x_3^2 + x_5^2} \\ \frac{x_1 x_2 + x_3 x_4 + x_5 x_6}{\sqrt{x_1^2 + x_3^2 + x_5^2}} \\ \arctan \frac{x_5}{x_1} \\ \arccos \frac{x_3}{\sqrt{x_1^2 + x_3^2 + x_5^2}} \end{bmatrix} + \begin{bmatrix} N(0, (1)^2) \\ N(0, (10)^2) \\ N(0, (10^{-3})^2) \\ N(0, (10^{-1})^2) \end{bmatrix} \tag{17}$$

With both the system dynamics and measurements model, the Propagation Step can be defined.

3. Propagation

The propagation step involves two integrations over the time step. The first integration is:

$$\dot{\hat{\boldsymbol{x}}}(\boldsymbol{t}) = \boldsymbol{f}(\hat{\boldsymbol{x}}(\boldsymbol{t}), t) \tag{18}$$

Where:

$\hat{\boldsymbol{x}}\boldsymbol{(t)}$ = Estimated state.

This uses the integration defined in equations (5), (6), and (7). The second equation is:

$$\dot{\boldsymbol{P}}\boldsymbol{(t)} = F(\hat{\boldsymbol{x}}\boldsymbol{(t)})P(t) + P(t)F^T(\hat{\boldsymbol{x}}\boldsymbol{(t)}) + G(t)Q_s(T)G^T(t) \tag{19}$$

Where:

F = Jacobian of system dynamics.

P = Covariance Matrix.

Q_s = Power Spectral Density.

The Power Spectral Density is:

$$Q_s = \begin{bmatrix} (1)^2 & 0 & 0 \\ 0 & (10^{-1})^2 & 0 \\ 0 & 0 & (1)^2 \end{bmatrix} \tag{20}$$

From the process noise distributions. The gain step is next.

4. Gain

The 3 gain parameters are:

$$W_k = H_k(\hat{\boldsymbol{x}}_k^-)P_k^-H_k^T(\hat{\boldsymbol{x}}_k^-) + R_k \tag{21}$$

$$C_k = P_k^-H_k^T(\hat{\boldsymbol{x}}_k^-) \tag{22}$$

$$K_k = C_kW_k^{-1} \tag{23}$$

Where:

W_k = Innovations Covariance.

C_k = Cross-Covariance.

K_k = Kalman Gain.

H = Jacobian of Measurement Model.

P^- = A Priori Covariance Matrix.

R_k = Measurement Noise Covariance Matrix.

The measurement noise covariance matrix is defined as:

$$R_k = \begin{bmatrix} (1)^2 & 0 & 0 & 0 \\ 0 & (10)^2 & 0 & 0 \\ 0 & 0 & (10^{-3})^2 & 0 \\ 0 & 0 & 0 & (10^{-1})^2 \end{bmatrix} \tag{24}$$

Finally, the update step of the EKF.

5. Update

The update is defined by 3 equations:

$$\hat{\boldsymbol{z}}_{\boldsymbol{k}} = \boldsymbol{h}_{\boldsymbol{k}}(\hat{\boldsymbol{x}}_{\boldsymbol{k}}) \tag{25}$$

$$\hat{\boldsymbol{x}}^{+}_{\boldsymbol{k}} = \hat{\boldsymbol{x}}^{-}_{\boldsymbol{k}} + K_k(z_k - \hat{\boldsymbol{z}}_{\boldsymbol{k}}) \tag{26}$$

$$P^{+}_{k} = P^{-}_{k} - C_k K^T_k - K_k C^T_k + K_k W_k K^T_k \tag{27}$$

Where:

$\hat{\boldsymbol{z}}_{\boldsymbol{k}}$ = A Posteriori Measurement Estimate.

$\hat{\boldsymbol{x}}^{+}_{\boldsymbol{k}}$ = A Posteriori Estimate.

P^{+}_{k} = A Posteriori Covariance Matrix.

The EKF needs to initialized in order for it to estimate the trajectory of the ball.

6. Initialization

According to SwingManGolf ([12]), the average long shot in Golf has an initial speed of 198 ft/s. If it assumed that the golfer shoots a relatively straight shot from the tee (origin) with moderate inclination, $\theta = 4\,\text{deg}$ and $\phi = 60\,\text{deg}$. Therefore, the initial estimate is:

$$\hat{\boldsymbol{x}}_{\boldsymbol{0}} = \begin{bmatrix} 0 \\ 171 \\ 0 \\ 99 \\ 0 \\ 11.96 \end{bmatrix} \tag{28}$$

The initial covariance matrix is then:

$$P_0 = E\left[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T\right] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 81 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 81 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3.8416 \end{bmatrix} \tag{29}$$

The EKF can now be implemented into MatLab and 5 cases analyzed. The 5 cases are:

1) Continuous measurements at every time step.
2) Measurement received every 0.01 seconds.
3) Measurement received every 0.1 seconds.
4) Measurements received continuously for first 1 second of flight.
5) Measurements received every 1/4600 seconds of the first 1 second. In line with FPS of TrackMan sensors. ([2]).

These cases are chosen to test the effectiveness of the EKF in predicting the final state of the golf ball under different measurement availability criteria, possible decreasing cost and computational rigor in real life scenarios.

IV. Results and Discussion

This section will outline the results and finding of each case. Each case utilized the same initial conditions and filtering technique. The singular difference in all cases is the availability of measurements.

A. Case 1

This case receives a measurement at every instant in time. Therefore, innovations and gains are calculated and an update is made at every time step. With every predication being updated after propagation, it is expected that this case results in the most accurate tracing of the trajectory of the ball. It is also expected that the errors in coordinates and velocities is least in this case. This case should resemble the increased accuracy of the TrackMan as it performs similarly to the TrackMan system.

As demonstrated in figure 3, the EKF does very well to trace the trajectory of the golf ball over the course of its flight. The EKF and true state are almost indistinguishable at every time step. This result is expected as a measurement is provided at every time step to update the prediction.

These error graphs demonstrate the error in the a priori and a posteriori estimates of the state with sigma bounds plotted to ensure a satisfactory result. The error bounds often break the $\pm\sigma$ bounds, but rarely break the $\pm 3\sigma$ bounds. This indicates the EKF performed well. The bounds for the x-coordinate estimate widens at first, then narrows. This

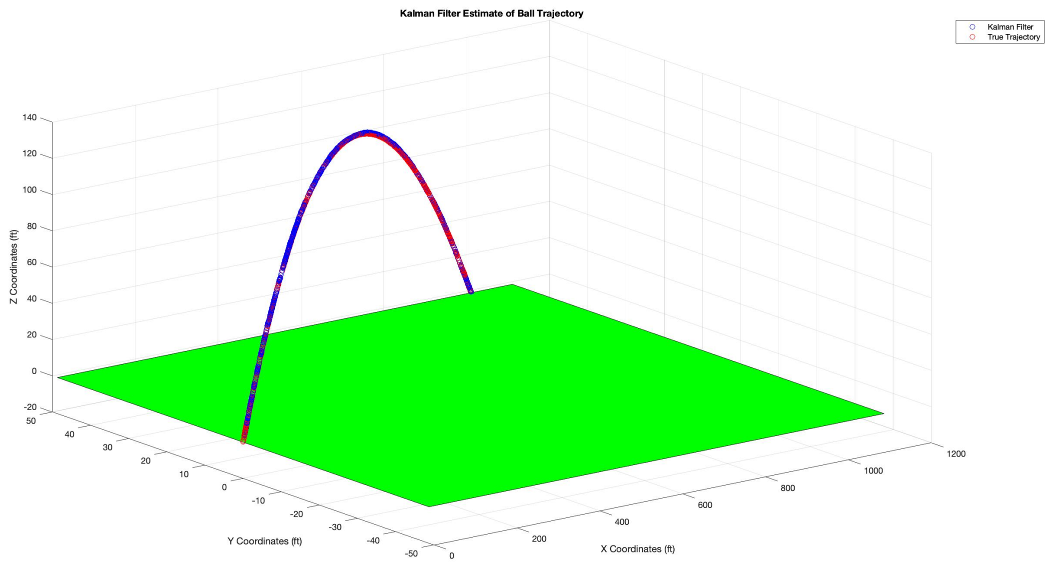

Fig. 3 Trajectory Tracing For Case 1.

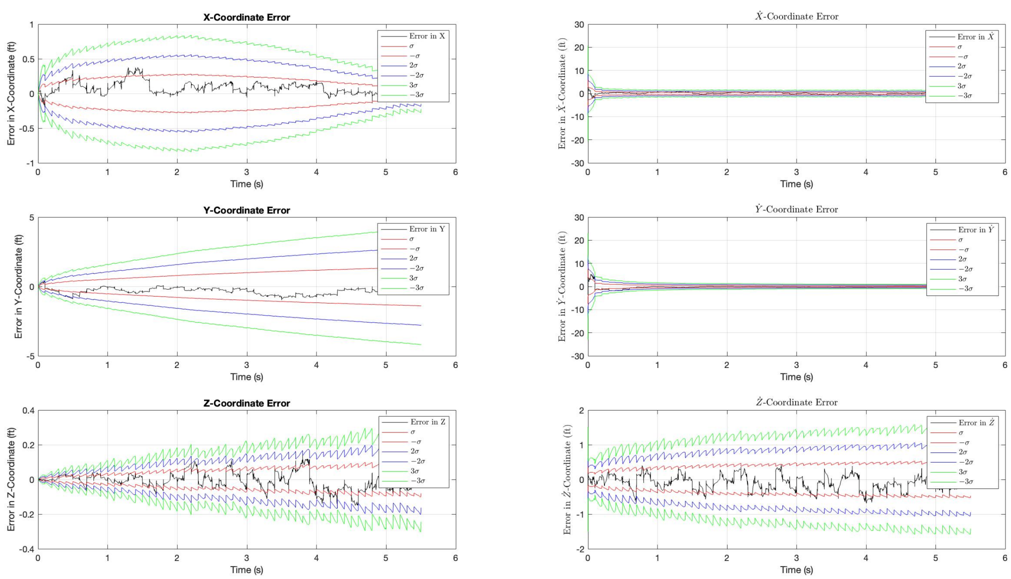

Fig. 4 Error Analysis for Case 1.

indicates that over time the uncertainty in the X position grows, and then begins to become more certain as time goes on. The Y and Z coordinates do not demonstrate a similar trend. Y and Z bounds grew, indicating their uncertainties grow over time and are greatest towards the end of the flight. A similar trend is seen in the Z component of velocity, which displays the greatest uncertainty at the end of the flight. X and Y components of velocity quickly narrowed and remained narrow for the remainder of the flight. Their uncertainties were least at the end of the flight. There are many reasons why uncertainties could be growing, one reason may be that the linearization of the dynamics and measurements may begin to fail towards the end of the flight. The filter could also be diverging towards the end of the trajectory, which could be caused by numerical instability in the propagation. Overall, the EKF performed very well when measurements are available at every time step.

B. Case 2

This case receives a measurement every 0.01 seconds. This is similar to a camera with a frame-per-second of 100 FPS. This is better than most commercial cameras, but poor for tracking applications. It is expected that this case performs worse than case 1 as an update is not made at every instant, meaning the EKF propagates forward the last updated prediction until it receives a new measurement.

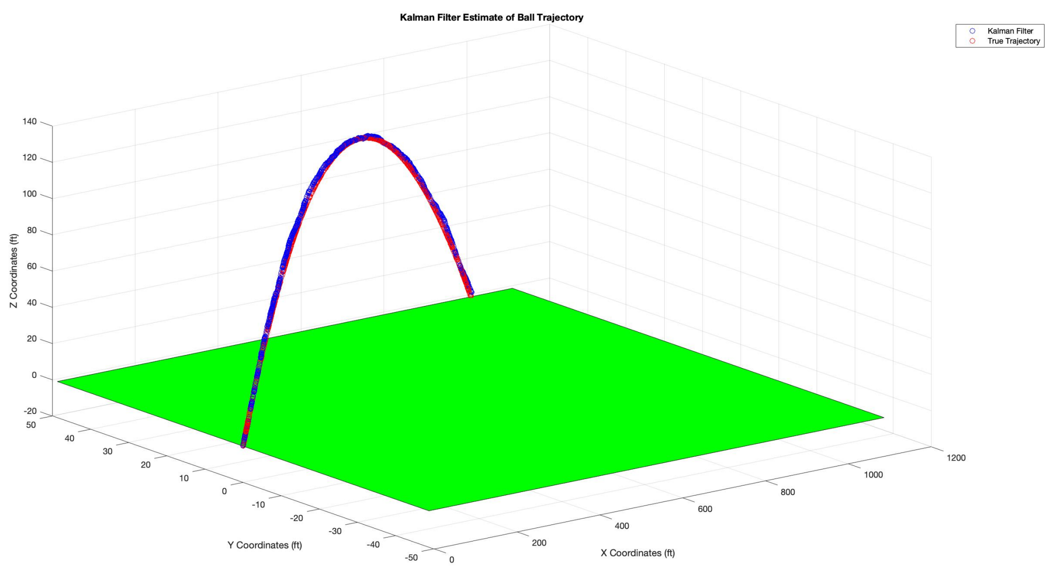

Fig. 5 Trajectory Tracing For Case 2.

Figure 5 displays relatively good tracing of the trajectory of the golf ball, indicating the model is able to accurately propagate without new measurements during short intervals of time.

The general shape of all graphs are similar to those of figure 4, indicating this case follows the same trends in uncertainty evolution. However, these bounds for this graph are larger, meaning there is greater uncertainity for this case than in case 1. This is expected as less measurements are provided to the EKF, so it can't update its predictions to be more accurate at every time step. The errors are all within the the $\pm 3\sigma$ bounds, indicating the estimation is fairly good and the errors are within the acceptable uncertainty.

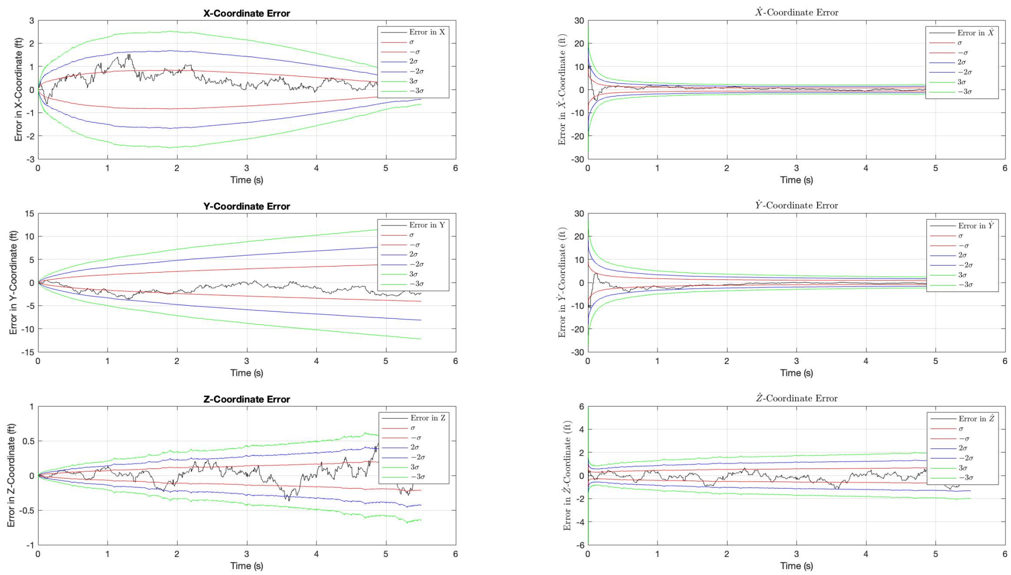

Fig. 6 Error Analysis for Case 2.

C. Case 3

A measurement is received once every 0.1 seconds. The trajectory chosen is in flight for less than 6 seconds, meaning a maximum of 60 observations of the entire flight. This case is expected to have discontinuities every time it updates as a results of the propagation neglecting process noise. This case is analogous with a camera with 10 FPS. This is extremely poor for cameras and especially for tracking applications.

As expected, there are many discontinuities over the flight. The trace has a sporadic change in 3 dimensional position every time a measurement is provided. This indicates the propagation of the estimate fails if left without an update for too long.

These graphs conserve the same general shape as the previous error graphs, indicating the filter is behaving in a similar manner as the trajectory evolves. The errors in almost all states touch or break the $\pm 3\sigma$ bounds during the initial stages of flight, indicating the filter is poorly estimating the states. As time evolves, errors are somewhat minimized as more updates are made to the trajectory. This means the filter converges to the true state as more measurements are provided, and therefore errors become minimized towards the end of the trajectory. Another possible reason for convergences is the elimination of initial state uncertainty, which is removed as more measurements are provided.

D. Case 4

This case estimates the trajectory after receiving measurements continuously for the first second. The previous cases demonstrated the inability of the propagation to accurately predict the next state if the time step is large. Therefore, it is expected this case will be the least accurate.

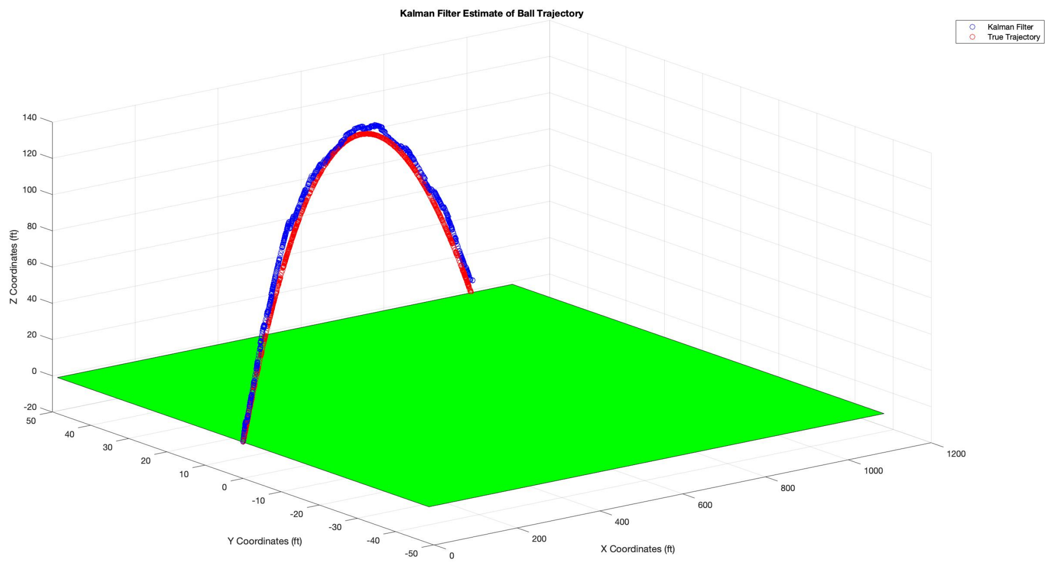

Fig. 7 Trajectory Analysis for Case 3.

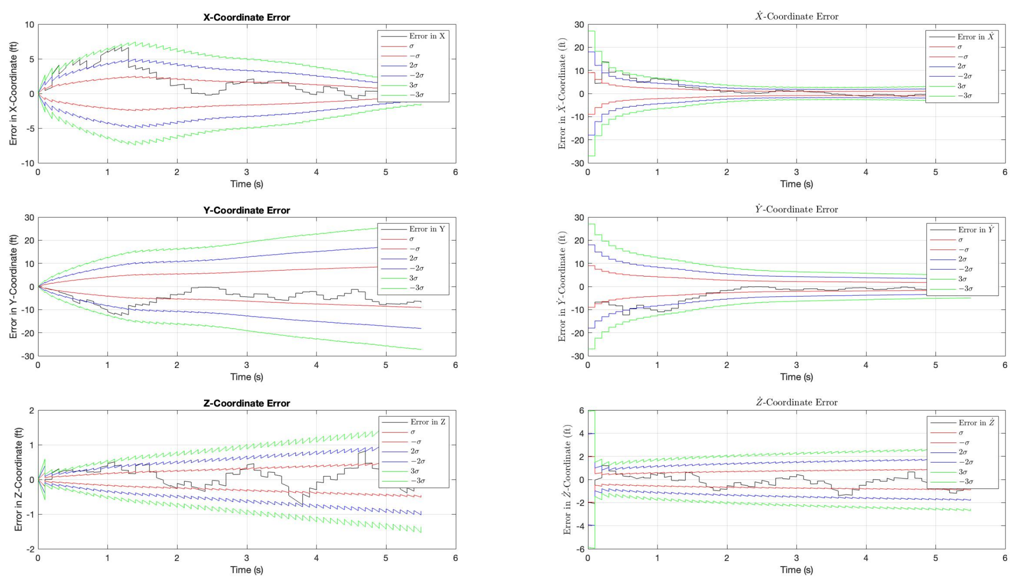

Fig. 8 Error Analysis for Case 3.

Figure 9 displays no discontinuities due to the initial continuous updates for the first 1 second, but then the EKF propagatesforward based on the final state update to the final state. That is why the trajectory of the flight begins to diverge away from the trajectory of the true state. The error graphs demonstrate this divergence. All states begin at similar errors to those of the previous cases, and then begin to increase steadily as the flight evolves. Final errors are much larger than previous cases. However, the σ bounds evolve with the error due to the covariance matrix estimation

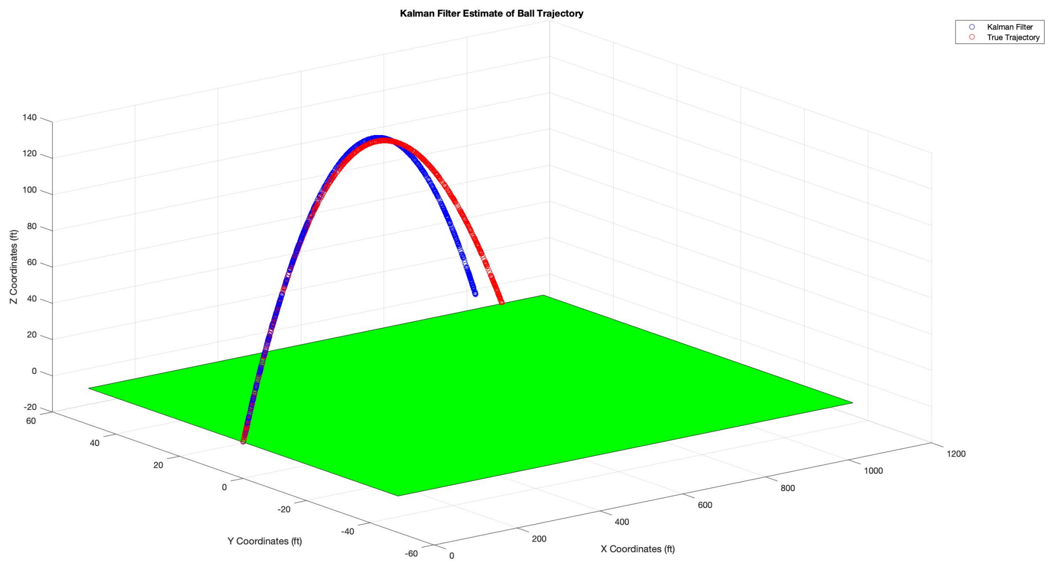

Fig. 9 Trajectory Analysis for Case 4.

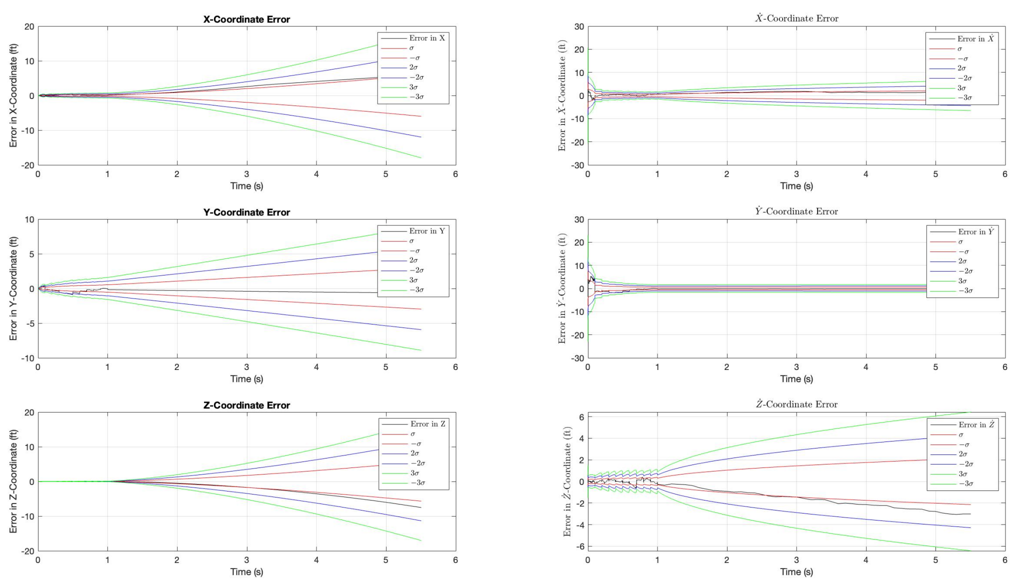

Fig. 10 Error Analysis for Case 4.

also growing in uncertainty. These errors remain within the $\pm 3\sigma$ bounds indicate the filter performs well.

E. Case 5

This case produces the same results as in case 4. Accurate estimates over the first second of flight, but then the trajectory diverges as time evolves. Errors in case 5 are similar to those in case 4. This indicates the FPS rate is able

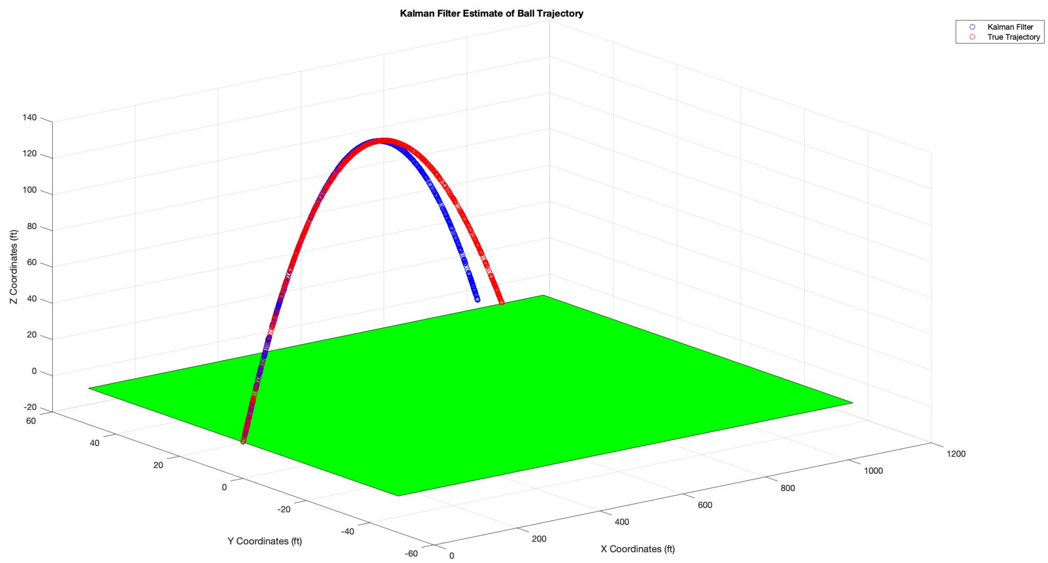

Fig. 11 Trajectory Analysis for Case 5.

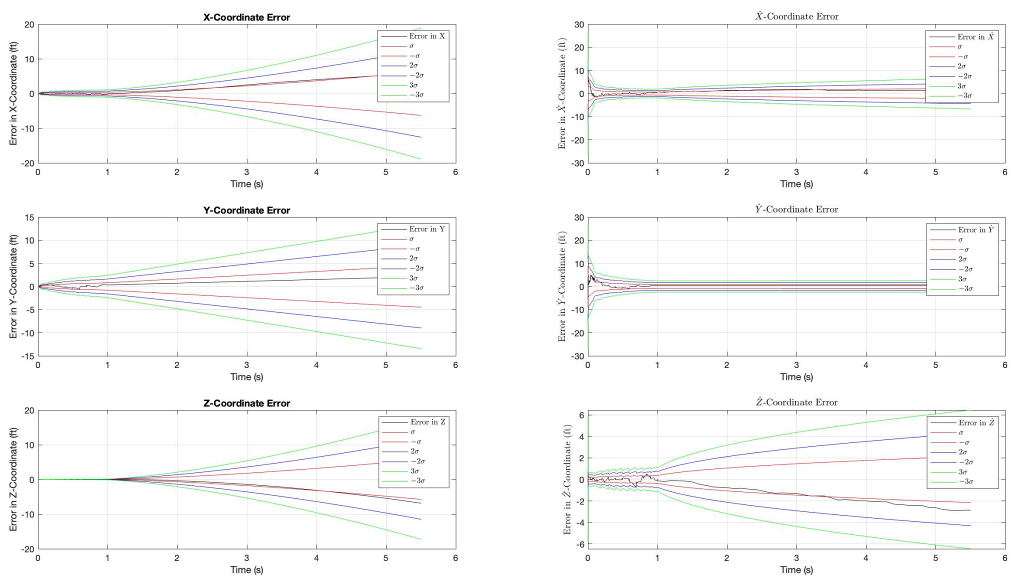

Fig. 12 Error Analysis for Case 5.

to capture enough measurements that the model does not need to propagate over large time steps causing it diverge. Therefore when the EKF propagates the states forward from the final update, it does not differ from if measurements are received continuously.

All 5 cases demonstrate implications and use cases in the sport of golf.

F. Limitations

The results of this study should be considered alongside the limitations. 2 main limitations effect the results of this study.

1. Limitation 1: Trajectory of golf ball follows the proposed model.

The model used to propagate states forward make many assumptions. It assumes no affects of wind, air pressure, ball type, surface type, and varying spin rate as the trajectory of the ball evolves over time. In reality, this is a very simplified version of the true trajectory of a golf ball. Also, the model assumes spin rates and Magnus are known and constant over the trajectory. This is not true as these values change over time and are dependent on the golf swing follow through, point of impact, and flight conditions. Real data should be fitted to the model to assess validity.

2. Limitation 2: Camera and Radar located at origin.

This model assumes that the camera and radar are co-located at the origin, where the ball is also located initially. In reality, almost no real systems use this set-up as this requires a camera and sensor per bay. Most systems use a few sensors to track the trajectory of multiple balls at a time. A difficult but interesting tracking problem to be explored.

V. Conclusion

In all, it was found the Kalman performs well enough to predict the final state within a certain range of error, but does not perform well enough to continuously trace the trajectory of the ball for all cases. The findings of each case can be summarized into implications.

A. Implications

All cases are applicable to different experiences and use cases of golf.

1. Case 1

Case 1, similar to the TrackMan systems used, are best suited for instances when the entire trajectory of the flight needs to be accurately modeled with little error. The continuous availability of measurements means the EKF is able to continuously update its estimate and remove any errors as time evolves. Even if the initial speed of the ball isn't known, which is usually the case, the trace of the trajectory is extremely accurate. However, for a system that implements this process, high computational costs will be associated. On a very small scale, this method took the longest run time of all other cases with 7.7 seconds. High costs could be incurred when attempting to have this process occur in real time, as seen by the high purchase cost of the TrackMan.

2. Case 2

This case is suitable for more casual use applications of tracers, such as drive ranges and top golf. The exact evolution of the trajectory is not as crucial as it may be in professional golf tournaments, but it still provides players with a more immerse experience. Also, this method is less computationally extensive due to the less number of measurements taken with a run time of 6.6 seconds. This means that the cameras used for this method are not required to be as high performing as those used in case 1, indicating less costs incurred.

3. Case 3

This case should be employed in instances where tracing the trajectory is not of use. The final state of the ball is roughly estimated well, and therefore this can be used for ranges that give users the opportunity to play virtual games that only require the final state of the golf ball. However, traceability is not applicable for the method. The EKF relies too heavily on propagation and does not update enough times to become more accurate. This method is the least computationally extensive (6.5 seconds) and can be employed with fairly underwhelming camera technology (10 FPS). This case proves the update and high-performing update and correction step, indicating a well selected measurement noise covariance matrix and spectral density matrix, leading to good innovations and cross-s used to calculate Kalman gain.

4. Case 4 and 5

These cases should only be used when a rough estimate of final state is important as these methods to not produce accurate tracing as a result of the few measurements provided to the EKF relative to the entire flight. External factors affect the trajectory of the ball in flight and the propagation step alone does not account for these uncertainties. These methods, however, would be the least costly and computationally extensive as measurements and updates only occur in the first 1 second.

B. Focus of Future Work

This study provides a preliminary analysis to a number of follow-up investigations. To emulate real life systems, multi-object multi-sensor tracking should be explored for these applications. The purpose of the study would be to repeat the measurement availability study for multi-object multi-sensor tracking, assessing how a different filtering technique is able to predict the trajectory and final states with a similar problem. This technique should also measure spin rates and Magnus effects, and update states this way. This should be done by also including the swing employed by the golfer. This study should also be extended to include real data from local driving ranges. This would assess the validity of the proposed dynamics and measurement models. Also, using real data to supply measurements to the EKF will assess its performance in a more realistic setting. It is hypothesized that the EKF will not perform as well when provided real data due to the large number of assumptions made in the formulation of the models, such as the assumption that some external factors are negligible. If using real data, nonlinear least squares method should be used to back out the initial state and covariance matrix after a few measurements are collected. Due to the nonlinear nature of the system dynamics, it is hypothesized that an Unscented Kalman Filter (UKF) would perform better due to its ability to directly approximate the mean and covariance of a target distribution. An attempt was made to create a UKF, however the program failed as the covariance matrix is unable to remain positive definite as time evolves. This indicates that the covariance matrix is no longer an accurate representation of the state uncertainties. This could be caused by numerical instability in the integration during propagation caused rounding off errors. This could also be due to the nonlinear nature of the dynamics, although UKF's are meant to handle nonlinear systems better than EKF's. Another possibility is that the noise of the system is causing sporadic updates, distorting the covariance matrix. A possibility could be to write the UKF as a Square Root Unscented Kalman Filter, which would provide better numerical stability and accuracy [13].

Appendix

Reflections By The Author

Overall, this was a fun and exciting project. The initial proposal was filled with optimism on the ability to acquire real data. Real data was never received and this caused a great shift in scope. A financial Kalman filter was explored, however with the time remaining and the model complexity it was decided to continue with the golf topic. I did the best I could with the time I had left. I initially began with little passion with the exploration of the topic, but as the filter was built, and more research was done on the applications, it became more and more exciting. Knowing what I know now, I would explore multi-object multi-sensor tracking for this project because that is more realistic. A better model would have been derived, with less assumptions and more characteristics of flight measured. I would have continued to look for real data, because that would have made the project much better. In all, this was a fun project that I wish I did more with. I hope to continue to explore the use of filtering techniques on topics I have more passion for. Very grateful for the class and for this experience. #nofilter

References

[1] PGatour, "PGA TOUR selects TrackMan tracking and tracing solution beginning in 2022," , 2022. URL `https://www.pgatour.com/article/news/latest/2022/02/02/pga-tour-selects-trackman-tracking-tracing-solution-beginning-in-2022`, [Accessed: 05-12-2023].

[2] TrackMan, "-," , -. URL `https://www.trackman.com/golf/launch-monitors/tech-specs`, [Accessed: 19-11-2023].

[3] TopTracer, "-," , -. URL `https://toptracer.com/`, [Accessed: 18-11-2023].

[4] Forsgren, D., "System and Method for Three Dimensional Object Tracking Using Combination of Radar and Image Data," , ????

[5] Burglund, B., and Street, R., "Golf Ball Flight Dynamics," Tech. rep., -, 2011.

[6] LeGrand, K., "Chapter 4: Kalman Filtering," , ????.

[7] LeGrand, K., "Chapter 4: Kalman Filtering," , ????.

[8] LeGrand, K., "4.2: Continuous-Discrete Kalman Filtering," , ????.

[9] LeGrand, K., "4.4: Continuous-Discrete Extended Kalman Filtering," , ????.

[10] Jiao, L., Pan, Q., Liang, Y., and Yang, F., "A Nonlinear Tracking Algorithm with Range-rate Measurements Based on Unbaised Measurement Conversion," Tech. rep., Northwestern Polytechnical University, -.

[11] BlackBoxGolf, "Are TrackmMan Golf Siumulators Distances Accurate?" , 2023. URL `https://www.blackboxgolf.es/blog/are-trackman-golf-simulators-distances-accurate`, [Accessed: 05-12-2023].

[12] "Average Golf Swing Speed Chart," , 2022. URL `https://swingmangolf.com/average-golf-swing-speed-chart-2/`, [Accessed: 11-24-2023].

[13] Holton, G. A., *Non-Positive Definite Covariance Matrix*, Online, 2013, Chap. 7.

[14] Peter E. Jenkins*, M. R. M. S., Joseph Arellano, "Drag Coefficients of Golf Balls," *World Journal of Mechanics*, ????

[15] Wall, J., "These golf balls recorded the highest spin rates during our robot testing," *Golf.com*, 2023. URL `https://golf.com/gear/golf-balls/golf-ball-2023-club-test-robot-testing/`.

[16] A. Kharlamov, P. V., Z. Chara, "Magnus and Drag Forces Acting on Golf Ball," ????

Programs Used

EKFCase1

```
%% AAE590ET Project: Case 1
clear
clc
close all
rng('Default')
tic
% System Modeling
w = [1, 10^-1, 1]; % Process noise definition (standard deviations)
Q = diag([w(1)^2, w(2)^2, w(3)^2]); % (covariances)
G = [0, 0, 0; 1, 0, 0; 0, 0, 0; 0, 1, 0; 0, 0, 0; 0, 0, 1]; % Mapping Matrix
```

```
timespan = 0:0.1:500; % Defining timespan for noise
v = [1, 10, 10^-3, 10^-1]; % Measurement noise definition (standard deviations)
R = diag(v.^2);
tspan = 0:0.1:100;
x0 = [0, 180, 0, 90, 0, 10];

[noise] = getNoise(w, timespan); % Creating noise array
[Flight, Time] = getTrajectory(noise, G, tspan, x0); % System Dynamics Model



% Measurement Model
[range, range_rate, theta, phi] = getMeasurements(Flight, v);
[cart_traj] = getCartesian(range, theta, phi);

% Storage Setup
xkm_history = NaN(6, length(Time));
xkp_history = NaN(6, length(Time));
Pkm_history = NaN(6, 6, length(Time));
Pkp_history = NaN(6, 6, length(Time));


% Initial Conditions
x0_hat = [0, 171, 0, 99, 0, 11.96];
P0 = diag((x0-x0_hat).*(x0-x0_hat));
init_conditions = [x0_hat(:); P0(:)];
xkm_pr = x0_hat;
Pkm_pr = P0;

% Kalman Filtering
options = odeset('RelTol', 1e-10, 'AbsTol', 1e-10);
```

```
tkm = 0;
zk = [range; range_rate; theta; phi];
for i = 2:length(Time)

    % Priori Prediction
    tk = Time(i);
    span = [tkm, tk];
    init_conditions = [xkm_pr(:); Pkm_pr(:)];
    [t, propagate] = ode45(@(t, propagate) Propagate(t, propagate, G, Q), span
        , init_conditions); % This is right
    xkm = propagate(end, 1:6).';
    Pkm = reshape(propagate(end, 7:end), 6, 6);

    % Innovations wrong
    H = getMeasurementJacobian(xkm);
    Wk = H*Pkm*H.' + R;
    Ck = Pkm*H.';

    % Gain wrong
    Kk = Ck / Wk;

    % Update
    z_est = getEstimate(xkm)';
    Zk = zk(:, i); %zk(:, find(Time <= tk, 1, 'last'));
    xkp = xkm + Kk * (zk(:, find(Time <= tk, 1, 'last')) - z_est);
    Pkp = Pkm - Ck*Kk.' - Kk*Ck.' + Kk*Wk*Kk.';
    Pkp = 0.5 * (Pkp + Pkp.');

    % Storage
    xkm_history(:, i) = xkm;
    xkp_history(:, i) = xkp;
```

```
    Pkm_history(:,:, i) = Pkm;
    Pkp_history(:,:,i) = Pkp;

    % Recursive
    xkm_pr = xkp;
    Pkm_pr = Pkp;
    tkm = tk;
end

figure
plot3(xkp_history(1, :), xkp_history(5, :), xkp_history(3, :), 'bo')
hold on
plot3(Flight(1, :), Flight(5, :), Flight(3, :), 'ro')
fill3([0, max(Flight(1, :))*1.1, max(Flight(1, :))*1.1, 0], [-max(Flight(5, :)), -max(Flight(5, :)), max(Flight(5, :)), max(Flight(5, :))], [0, 0, 0, 0], 'g')
grid on
xlabel('X Coordinates (ft)')
ylabel('Y Coordinates (ft)')
zlabel('Z Coordinates (ft)')
title('Kalman Filter Estimate of Ball Trajectory')
legend('Kalman Filter', 'True Trajectory')

%% Sigma Bounds
sigma_1p = sqrt(squeeze(Pkp_history(1,1,:)));
sigma_1m = sqrt(squeeze(Pkm_history(1,1,:)));
sigma_2p = sqrt(squeeze(Pkp_history(2,2,:)));
sigma_2m = sqrt(squeeze(Pkm_history(2,2,:)));
sigma_3p = sqrt(squeeze(Pkp_history(3,3,:)));
sigma_3m = sqrt(squeeze(Pkm_history(3,3,:)));
sigma_4p = sqrt(squeeze(Pkp_history(4,4,:)));
sigma_4m = sqrt(squeeze(Pkm_history(4,4,:)));
```

```
sigma_5p = sqrt(squeeze(Pkp_history(5,5,:)));
sigma_5m = sqrt(squeeze(Pkm_history(5,5,:)));
sigma_6p = sqrt(squeeze(Pkp_history(6,6,:)));
sigma_6m = sqrt(squeeze(Pkm_history(6,6,:)));

%% Errors
error1p = Flight(1, :) - xkp_history(1,:);
error1m = Flight(1, :) - xkm_history(1,:);
error2p = Flight(2, :) - xkp_history(2,:);
error2m = Flight(2, :) - xkm_history(2,:);
error3p = Flight(3, :) - xkp_history(3,:);
error3m = Flight(3, :) - xkm_history(3,:);
error4p = Flight(4, :) - xkp_history(4,:);
error4m = Flight(4, :) - xkm_history(4,:);
error5p = Flight(5, :) - xkp_history(5,:);
error5m = Flight(5, :) - xkm_history(5,:);
error6p = Flight(6, :) - xkp_history(6,:);
error6m = Flight(6, :) - xkm_history(6,:);

%% Plot Estimation Error and Associated  3  Sigma Bounds
% Each State
figure
subplot(3, 2, 1)
plot(Time, error1p, 'k')
hold on
plot(Time, 1 * sigma_1p, 'r')
plot(Time, -1 * sigma_1p, 'r')
plot(Time, 2 * sigma_1p, 'b')
plot(Time, -2 * sigma_1p, 'b')
plot(Time, 3 * sigma_1p, 'g')
plot(Time, -3 * sigma_1p, 'g')
grid on
```

```
xlabel('Time (s)')
ylabel('Error in X-Coordinate (ft)')
title('X-Coordinate Error')
legend('Error in X', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
    sigma$', '$-3\sigma$',  'Interpreter', 'latex' )


subplot(3, 2, 2)
plot(Time, error2p, 'k')
hold on
plot(Time, 1 * sigma_2p, 'r')
plot(Time, -1 * sigma_2p, 'r')
plot(Time, 2 * sigma_2p, 'b')
plot(Time, -2 * sigma_2p, 'b')
plot(Time, 3 * sigma_2p, 'g')
plot(Time, -3 * sigma_2p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{X}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{X}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{X}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$
    ', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )


subplot(3, 2, 3)
plot(Time, error3p, 'k')
hold on
plot(Time, 1 * sigma_3p, 'r')
plot(Time, -1 * sigma_3p, 'r')
plot(Time, 2 * sigma_3p, 'b')
plot(Time, -2 * sigma_3p, 'b')
plot(Time, 3 * sigma_3p, 'g')
```

```
plot(Time, -3 * sigma_3p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in Y-Coordinate (ft)')
title('Y-Coordinate Error')
legend('Error in Y', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
    sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 4)
plot(Time, error4p, 'k')
hold on
plot(Time, 1 * sigma_4p, 'r')
plot(Time, -1 * sigma_4p, 'r')
plot(Time, 2 * sigma_4p, 'b')
plot(Time, -2 * sigma_4p, 'b')
plot(Time, 3 * sigma_4p, 'g')
plot(Time, -3 * sigma_4p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{Y}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{Y}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{Y}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$
    ', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 5)
plot(Time, error5p, 'k')
hold on
plot(Time, 1 * sigma_5p, 'r')
plot(Time, -1 * sigma_5p, 'r')
plot(Time, 2 * sigma_5p, 'b')
plot(Time, -2 * sigma_5p, 'b')
plot(Time, 3 * sigma_5p, 'g')
```

```matlab
plot(Time, -3 * sigma_5p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in Z-Coordinate (ft)')
title('Z-Coordinate Error')
legend('Error in Z', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 6)
plot(Time, error6p, 'k')
hold on
plot(Time, 1 * sigma_6p, 'r')
plot(Time, -1 * sigma_6p, 'r')
plot(Time, 2 * sigma_6p, 'b')
plot(Time, -2 * sigma_6p, 'b')
plot(Time, 3 * sigma_6p, 'g')
plot(Time, -3 * sigma_6p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{Z}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{Z}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{Z}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )
toc
```

```matlab
%% AAE590ET Project: Case 2
clear
clc
close all
rng('Default')
tic
% System Modeling
w = [1, 10^-1, 1]; % Process noise definition (standard deviations)
```

```
Q = diag([w(1)^2, w(2)^2, w(3)^2]); % (covariances)
G = [0, 0, 0; 1, 0, 0; 0, 0, 0; 0, 1, 0; 0, 0, 0; 0, 0, 1]; % Mapping Matrix
timespan = 0:0.1:500; % Defining timespan for noise
v = [1, 10, 10^-3, 10^-1]; % Measurement noise definition (standard deviations
    )
R = diag(v.^2);
tspan = 0:0.1:100;
x0 = [0, 180, 0, 90, 0, 10];

[noise] = getNoise(w, timespan); % Creating noise array
[Flight, Time] = getTrajectory(noise, G, tspan, x0); % System Dynamics Model


% Measurement Model
[range, range_rate, theta, phi] = getMeasurements(Flight, v);
[cart_traj] = getCartesian(range, theta, phi);

% Storage Setup
xkm_history = NaN(6, length(Time));
xkp_history = NaN(6, length(Time));
Pkm_history = NaN(6, 6, length(Time));
Pkp_history = NaN(6, 6, length(Time));


% Initial Conditions
x0_hat = [0, 171, 0, 99, 0, 11.96];
P0 = diag((x0-x0_hat).*(x0-x0_hat));
init_conditions = [x0_hat(:); P0(:)];
xkm_pr = x0_hat;
Pkm_pr = P0;
tkm = 0;
options = odeset('RelTol', 1e-10, 'AbsTol', 1e-10);
```

```

% Measurement Availability
settime = 0;
measurementstep = 0.01;
zk = NaN(4, length(Time));


while settime <= Time(end)
    instance = find(Time <= settime, 1, 'last');
    zk(:, instance) = [range(instance); range_rate(instance); theta(instance);
         phi(instance)];
    settime =  settime + measurementstep;

end


% Kalman Filtering
for i = 2:length(Time)

    % Priori Prediction
    tk = Time(i);
    span = [tkm, tk];
    init_conditions = [xkm_pr(:); Pkm_pr(:)];
    [t, propagate] = ode45(@(t, propagate) Propagate(t, propagate, G, Q), span
        , init_conditions); % This is right
    xkm = propagate(end, 1:6).';
    Pkm = reshape(propagate(end, 7:end), 6, 6);

    if isnan(zk(:, i)) == [0, 0, 0, 0];
    % Innovations
    H = getMeasurementJacobian(xkm);
    Wk = H*Pkm*H.' + R;
```

```
    Ck = Pkm*H.';

    % Gain
    Kk = Ck / Wk;

    % Update
    z_est = getEstimate(xkm)';
    Zk = zk(:, i);
    xkp = xkm + Kk * (zk(:, find(Time <= tk, 1, 'last')) - z_est);
    Pkp = Pkm - Ck*Kk.' - Kk*Ck.' + Kk*Wk*Kk.';
    Pkp = 0.5 * (Pkp + Pkp.');

    else
        xkp = xkm;
        Pkp = Pkm;

    end

    % Storage
    xkm_history(:, i) = xkm;
    xkp_history(:, i) = xkp;
    Pkm_history(:,:, i) = Pkm;
    Pkp_history(:,:,i) = Pkp;

    % Recursive
    xkm_pr = xkp;
    Pkm_pr = Pkp;
    tkm = tk;
end

figure
plot3(xkp_history(1, :), xkp_history(5, :), xkp_history(3, :), 'bo')
```

```
hold on
plot3(Flight(1, :), Flight(5, :), Flight(3, :), 'ro')
fill3([0, max(Flight(1, :))*1.1, max(Flight(1, :))*1.1, 0], [-max(Flight(5, :)), -max(Flight(5, :)), max(Flight(5, :)), max(Flight(5, :))], [0, 0, 0, 0], 'g')
grid on
xlabel('X Coordinates (ft)')
ylabel('Y Coordinates (ft)')
zlabel('Z Coordinates (ft)')
title('Kalman Filter Estimate of Ball Trajectory')
legend('Kalman Filter', 'True Trajectory')

%% Sigma Bounds
sigma_1p = sqrt(squeeze(Pkp_history(1,1,:)));
sigma_1m = sqrt(squeeze(Pkm_history(1,1,:)));
sigma_2p = sqrt(squeeze(Pkp_history(2,2,:)));
sigma_2m = sqrt(squeeze(Pkm_history(2,2,:)));
sigma_3p = sqrt(squeeze(Pkp_history(3,3,:)));
sigma_3m = sqrt(squeeze(Pkm_history(3,3,:)));
sigma_4p = sqrt(squeeze(Pkp_history(4,4,:)));
sigma_4m = sqrt(squeeze(Pkm_history(4,4,:)));
sigma_5p = sqrt(squeeze(Pkp_history(5,5,:)));
sigma_5m = sqrt(squeeze(Pkm_history(5,5,:)));
sigma_6p = sqrt(squeeze(Pkp_history(6,6,:)));
sigma_6m = sqrt(squeeze(Pkm_history(6,6,:)));

%% Errors
error1p = Flight(1, :) - xkp_history(1,:);
error1m = Flight(1, :) - xkm_history(1,:);
error2p = Flight(2, :) - xkp_history(2,:);
error2m = Flight(2, :) - xkm_history(2,:);
error3p = Flight(3, :) - xkp_history(3,:);
```

```
error3m = Flight(3, :) - xkm_history(3,:);
error4p = Flight(4, :) - xkp_history(4,:);
error4m = Flight(4, :) - xkm_history(4,:);
error5p = Flight(5, :) - xkp_history(5,:);
error5m = Flight(5, :) - xkm_history(5,:);
error6p = Flight(6, :) - xkp_history(6,:);
error6m = Flight(6, :) - xkm_history(6,:);

%% Plot Estimation Error and Associated  3  Sigma Bounds
% Each State
figure
subplot(3, 2, 1)
plot(Time, error1p, 'k')
hold on
plot(Time, 1 * sigma_1p, 'r')
plot(Time, -1 * sigma_1p, 'r')
plot(Time, 2 * sigma_1p, 'b')
plot(Time, -2 * sigma_1p, 'b')
plot(Time, 3 * sigma_1p, 'g')
plot(Time, -3 * sigma_1p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in X-Coordinate (ft)')
title('X-Coordinate Error')
legend('Error in X', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
    sigma$', '$-3\sigma$',  'Interpreter', 'latex' )


subplot(3, 2, 2)
plot(Time, error2p, 'k')
hold on
plot(Time, 1 * sigma_2p, 'r')
```

```
plot(Time, -1 * sigma_2p, 'r')
plot(Time, 2 * sigma_2p, 'b')
plot(Time, -2 * sigma_2p, 'b')
plot(Time, 3 * sigma_2p, 'g')
plot(Time, -3 * sigma_2p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{X}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{X}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{X}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$
    ', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )


subplot(3, 2, 3)
plot(Time, error3p, 'k')
hold on
plot(Time, 1 * sigma_3p, 'r')
plot(Time, -1 * sigma_3p, 'r')
plot(Time, 2 * sigma_3p, 'b')
plot(Time, -2 * sigma_3p, 'b')
plot(Time, 3 * sigma_3p, 'g')
plot(Time, -3 * sigma_3p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in Y-Coordinate (ft)')
title('Y-Coordinate Error')
legend('Error in Y', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
    sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 4)
plot(Time, error4p, 'k')
hold on
```

```
plot(Time, 1 * sigma_4p, 'r')
plot(Time, -1 * sigma_4p, 'r')
plot(Time, 2 * sigma_4p, 'b')
plot(Time, -2 * sigma_4p, 'b')
plot(Time, 3 * sigma_4p, 'g')
plot(Time, -3 * sigma_4p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{Y}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{Y}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{Y}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$
    ', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 5)
plot(Time, error5p, 'k')
hold on
plot(Time, 1 * sigma_5p, 'r')
plot(Time, -1 * sigma_5p, 'r')
plot(Time, 2 * sigma_5p, 'b')
plot(Time, -2 * sigma_5p, 'b')
plot(Time, 3 * sigma_5p, 'g')
plot(Time, -3 * sigma_5p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in Z-Coordinate (ft)')
title('Z-Coordinate Error')
legend('Error in Z', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
    sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 6)
plot(Time, error6p, 'k')
hold on
```

```
plot(Time, 1 * sigma_6p, 'r')
plot(Time, -1 * sigma_6p, 'r')
plot(Time, 2 * sigma_6p, 'b')
plot(Time, -2 * sigma_6p, 'b')
plot(Time, 3 * sigma_6p, 'g')
plot(Time, -3 * sigma_6p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{Z}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{Z}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{Z}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$
    ', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )
toc
```

```
%% AAE590ET Project: Case 3
clear
clc
close all
rng('Default')
tic
% System Modeling
w = [1, 10^-1, 1]; % Process noise definition (standard deviations)
Q = diag([w(1)^2, w(2)^2, w(3)^2]); % (covariances)
G = [0, 0, 0; 1, 0, 0; 0, 0, 0; 0, 1, 0; 0, 0, 0; 0, 0, 1]; % Mapping Matrix
timespan = 0:0.1:500; % Defining timespan for noise
v = [1, 10, 10^-3, 10^-1]; % Measurement noise definition (standard deviations
    )
R = diag(v.^2);
tspan = 0:0.1:100;
x0 = [0, 180, 0, 90, 0, 10];

[noise] = getNoise(w, timespan); % Creating noise array
[Flight, Time] = getTrajectory(noise, G, tspan, x0); % System Dynamics Model
```

```
% Measurement Model
[range, range_rate, theta, phi] = getMeasurements(Flight, v);
[cart_traj] = getCartesian(range, theta, phi);

% Storage Setup
xkm_history = NaN(6, length(Time));
xkp_history = NaN(6, length(Time));
Pkm_history = NaN(6, 6, length(Time));
Pkp_history = NaN(6, 6, length(Time));


% Initial Conditions
x0_hat = [0, 171, 0, 99, 0, 11.96];
P0 = diag((x0-x0_hat).*(x0-x0_hat));
init_conditions = [x0_hat(:); P0(:)];
xkm_pr = x0_hat;
Pkm_pr = P0;
tkm = 0;
options = odeset('RelTol', 1e-10, 'AbsTol', 1e-10);

% Measurement Availability
settime = 0;
measurementstep = 0.1;
zk = NaN(4, length(Time));


while settime <= Time(end)
    instance = find(Time <= settime, 1, 'last');
    zk(:, instance) = [range(instance); range_rate(instance); theta(instance);
        phi(instance)];
```

```
    settime =  settime + measurementstep;

end


% Kalman Filtering
for i = 2:length(Time)

    % Priori Prediction
    tk = Time(i);
    span = [tkm, tk];
    init_conditions = [xkm_pr(:); Pkm_pr(:)];
    [t, propagate] = ode45(@(t, propagate) Propagate(t, propagate, G, Q), span
        , init_conditions); % This is right
    xkm = propagate(end, 1:6).';
    Pkm = reshape(propagate(end, 7:end), 6, 6);

    if isnan(zk(:, i)) == [0, 0, 0, 0];
    % Innovations
    H = getMeasurementJacobian(xkm);
    Wk = H*Pkm*H.' + R;
    Ck = Pkm*H.';

    % Gain
    Kk = Ck / Wk;

    % Update
    z_est = getEstimate(xkm)';
    Zk = zk(:, i);
    xkp = xkm + Kk * (zk(:, find(Time <= tk, 1, 'last')) - z_est);
    Pkp = Pkm - Ck*Kk.' - Kk*Ck.' + Kk*Wk*Kk.';
    Pkp = 0.5 * (Pkp + Pkp.');
```

```
    else
        xkp = xkm;
        Pkp = Pkm;

    end

    % Storage
    xkm_history(:, i) = xkm;
    xkp_history(:, i) = xkp;
    Pkm_history(:,:, i) = Pkm;
    Pkp_history(:,:,i) = Pkp;

    % Recursive
    xkm_pr = xkp;
    Pkm_pr = Pkp;
    tkm = tk;
end

figure
plot3(xkp_history(1, :), xkp_history(5, :), xkp_history(3, :), 'bo')
hold on
plot3(Flight(1, :), Flight(5, :), Flight(3, :), 'ro')
fill3([0, max(Flight(1, :))*1.1, max(Flight(1, :))*1.1, 0], [-max(Flight(5, :)
    ), -max(Flight(5, :)), max(Flight(5, :)), max(Flight(5, :))], [0, 0, 0,
    0], 'g')
grid on
xlabel('X Coordinates (ft)')
ylabel('Y Coordinates (ft)')
zlabel('Z Coordinates (ft)')
title('Kalman Filter Estimate of Ball Trajectory')
legend('Kalman Filter', 'True Trajectory')
```

```
%% Sigma Bounds
sigma_1p = sqrt(squeeze(Pkp_history(1,1,:)));
sigma_1m = sqrt(squeeze(Pkm_history(1,1,:)));
sigma_2p = sqrt(squeeze(Pkp_history(2,2,:)));
sigma_2m = sqrt(squeeze(Pkm_history(2,2,:)));
sigma_3p = sqrt(squeeze(Pkp_history(3,3,:)));
sigma_3m = sqrt(squeeze(Pkm_history(3,3,:)));
sigma_4p = sqrt(squeeze(Pkp_history(4,4,:)));
sigma_4m = sqrt(squeeze(Pkm_history(4,4,:)));
sigma_5p = sqrt(squeeze(Pkp_history(5,5,:)));
sigma_5m = sqrt(squeeze(Pkm_history(5,5,:)));
sigma_6p = sqrt(squeeze(Pkp_history(6,6,:)));
sigma_6m = sqrt(squeeze(Pkm_history(6,6,:)));

%% Errors
error1p = Flight(1, :) - xkp_history(1,:);
error1m = Flight(1, :) - xkm_history(1,:);
error2p = Flight(2, :) - xkp_history(2,:);
error2m = Flight(2, :) - xkm_history(2,:);
error3p = Flight(3, :) - xkp_history(3,:);
error3m = Flight(3, :) - xkm_history(3,:);
error4p = Flight(4, :) - xkp_history(4,:);
error4m = Flight(4, :) - xkm_history(4,:);
error5p = Flight(5, :) - xkp_history(5,:);
error5m = Flight(5, :) - xkm_history(5,:);
error6p = Flight(6, :) - xkp_history(6,:);
error6m = Flight(6, :) - xkm_history(6,:);

%% Plot Estimation Error and Associated  3  Sigma Bounds
% Each State
figure
```

```
subplot(3, 2, 1)
plot(Time, error1p, 'k')
hold on
plot(Time, 1 * sigma_1p, 'r')
plot(Time, -1 * sigma_1p, 'r')
plot(Time, 2 * sigma_1p, 'b')
plot(Time, -2 * sigma_1p, 'b')
plot(Time, 3 * sigma_1p, 'g')
plot(Time, -3 * sigma_1p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in X-Coordinate (ft)')
title('X-Coordinate Error')
legend('Error in X', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
    sigma$', '$-3\sigma$',  'Interpreter', 'latex' )


subplot(3, 2, 2)
plot(Time, error2p, 'k')
hold on
plot(Time, 1 * sigma_2p, 'r')
plot(Time, -1 * sigma_2p, 'r')
plot(Time, 2 * sigma_2p, 'b')
plot(Time, -2 * sigma_2p, 'b')
plot(Time, 3 * sigma_2p, 'g')
plot(Time, -3 * sigma_2p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{X}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{X}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{X}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$
    ', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )
```

```
subplot(3, 2, 3)
plot(Time, error3p, 'k')
hold on
plot(Time, 1 * sigma_3p, 'r')
plot(Time, -1 * sigma_3p, 'r')
plot(Time, 2 * sigma_3p, 'b')
plot(Time, -2 * sigma_3p, 'b')
plot(Time, 3 * sigma_3p, 'g')
plot(Time, -3 * sigma_3p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in Y-Coordinate (ft)')
title('Y-Coordinate Error')
legend('Error in Y', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
    sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 4)
plot(Time, error4p, 'k')
hold on
plot(Time, 1 * sigma_4p, 'r')
plot(Time, -1 * sigma_4p, 'r')
plot(Time, 2 * sigma_4p, 'b')
plot(Time, -2 * sigma_4p, 'b')
plot(Time, 3 * sigma_4p, 'g')
plot(Time, -3 * sigma_4p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{Y}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{Y}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{Y}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$
```

```
', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 5)
plot(Time, error5p, 'k')
hold on
plot(Time, 1 * sigma_5p, 'r')
plot(Time, -1 * sigma_5p, 'r')
plot(Time, 2 * sigma_5p, 'b')
plot(Time, -2 * sigma_5p, 'b')
plot(Time, 3 * sigma_5p, 'g')
plot(Time, -3 * sigma_5p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in Z-Coordinate (ft)')
title('Z-Coordinate Error')
legend('Error in Z', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 6)
plot(Time, error6p, 'k')
hold on
plot(Time, 1 * sigma_6p, 'r')
plot(Time, -1 * sigma_6p, 'r')
plot(Time, 2 * sigma_6p, 'b')
plot(Time, -2 * sigma_6p, 'b')
plot(Time, 3 * sigma_6p, 'g')
plot(Time, -3 * sigma_6p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{Z}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{Z}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{Z}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$
```

```
', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )
toc
```

```
%% AAE590ET Project: Case 4
clear
clc
close all
rng('Default')
tic
% System Modeling
w = [1, 10^-1, 1]; % Process noise definition (standard deviations)
Q = diag([w(1)^2, w(2)^2, w(3)^2]); % (covariances)
G = [0, 0, 0; 1, 0, 0; 0, 0, 0; 0, 1, 0; 0, 0, 0; 0, 0, 1]; % Mapping Matrix
timespan = 0:0.1:500; % Defining timespan for noise
v = [1, 10, 10^-3, 10^-1]; % Measurement noise definition (standard deviations
    )
R = diag(v.^2);
tspan = 0:0.1:100;
x0 = [0, 180, 0, 90, 0, 10];

[noise] = getNoise(w, timespan); % Creating noise array
[Flight, Time] = getTrajectory(noise, G, tspan, x0); % System Dynamics Model


% Measurement Model
[range, range_rate, theta, phi] = getMeasurements(Flight, v);
[cart_traj] = getCartesian(range, theta, phi);

% Storage Setup
xkm_history = NaN(6, length(Time));
xkp_history = NaN(6, length(Time));
Pkm_history = NaN(6, 6, length(Time));
Pkp_history = NaN(6, 6, length(Time));
```

```
% Initial Conditions
x0_hat = [0, 171, 0, 99, 0, 11.96];
P0 = diag((x0-x0_hat).*(x0-x0_hat));
init_conditions = [x0_hat(:); P0(:)];
xkm_pr = x0_hat;
Pkm_pr = P0;
tkm = 0;
options = odeset('RelTol', 1e-10, 'AbsTol', 1e-10);

% Measurement Availability
maxtime = 1;
instance = find(Time <= maxtime);
zk = NaN(4, length(Time));

zk(:, instance) = [range(instance); range_rate(instance); theta(instance); phi
    (instance)];



% Kalman Filtering
for i = 2:length(Time)

    % Priori Prediction
    tk = Time(i);
    span = [tkm, tk];
    init_conditions = [xkm_pr(:); Pkm_pr(:)];
    [t, propagate] = ode45(@(t, propagate) Propagate(t, propagate, G, Q), span
        , init_conditions); % This is right
    xkm = propagate(end, 1:6).';
    Pkm = reshape(propagate(end, 7:end), 6, 6);
```

```
    if isnan(zk(:, i)) == [0, 0, 0, 0];
    % Innovations
    H = getMeasurementJacobian(xkm);
    Wk = H*Pkm*H.' + R;
    Ck = Pkm*H.';

    % Gain
    Kk = Ck / Wk;

    % Update
    z_est = getEstimate(xkm)';
    Zk = zk(:, i);
    xkp = xkm + Kk * (zk(:, find(Time <= tk, 1, 'last')) - z_est);
    Pkp = Pkm - Ck*Kk.' - Kk*Ck.' + Kk*Wk*Kk.';
    Pkp = 0.5 * (Pkp + Pkp.');

    else
        xkp = xkm;
        Pkp = Pkm;

    end

    % Storage
    xkm_history(:, i) = xkm;
    xkp_history(:, i) = xkp;
    Pkm_history(:,:, i) = Pkm;
    Pkp_history(:,:,i) = Pkp;

    % Recursive
    xkm_pr = xkp;
    Pkm_pr = Pkp;
```

```
    tkm = tk;
end

figure
plot3(xkp_history(1, :), xkp_history(5, :), xkp_history(3, :), 'bo')
hold on
plot3(Flight(1, :), Flight(5, :), Flight(3, :), 'ro')
fill3([0, max(Flight(1, :))*1.1, max(Flight(1, :))*1.1, 0], [-max(Flight(5, :)
    ), -max(Flight(5, :)), max(Flight(5, :)), max(Flight(5, :))], [0, 0, 0,
    0], 'g')
grid on
xlabel('X Coordinates (ft)')
ylabel('Y Coordinates (ft)')
zlabel('Z Coordinates (ft)')
title('Kalman Filter Estimate of Ball Trajectory')
legend('Kalman Filter', 'True Trajectory')

%% Sigma Bounds
sigma_1p = sqrt(squeeze(Pkp_history(1,1,:)));
sigma_1m = sqrt(squeeze(Pkm_history(1,1,:)));
sigma_2p = sqrt(squeeze(Pkp_history(2,2,:)));
sigma_2m = sqrt(squeeze(Pkm_history(2,2,:)));
sigma_3p = sqrt(squeeze(Pkp_history(3,3,:)));
sigma_3m = sqrt(squeeze(Pkm_history(3,3,:)));
sigma_4p = sqrt(squeeze(Pkp_history(4,4,:)));
sigma_4m = sqrt(squeeze(Pkm_history(4,4,:)));
sigma_5p = sqrt(squeeze(Pkp_history(5,5,:)));
sigma_5m = sqrt(squeeze(Pkm_history(5,5,:)));
sigma_6p = sqrt(squeeze(Pkp_history(6,6,:)));
sigma_6m = sqrt(squeeze(Pkm_history(6,6,:)));

%% Errors
```

```
error1p = Flight(1, :) - xkp_history(1,:);
error1m = Flight(1, :) - xkm_history(1,:);
error2p = Flight(2, :) - xkp_history(2,:);
error2m = Flight(2, :) - xkm_history(2,:);
error3p = Flight(3, :) - xkp_history(3,:);
error3m = Flight(3, :) - xkm_history(3,:);
error4p = Flight(4, :) - xkp_history(4,:);
error4m = Flight(4, :) - xkm_history(4,:);
error5p = Flight(5, :) - xkp_history(5,:);
error5m = Flight(5, :) - xkm_history(5,:);
error6p = Flight(6, :) - xkp_history(6,:);
error6m = Flight(6, :) - xkm_history(6,:);

%% Plot Estimation Error and Associated  3  Sigma Bounds
% Each State
figure
subplot(3, 2, 1)
plot(Time, error1p, 'k')
hold on
plot(Time, 1 * sigma_1p, 'r')
plot(Time, -1 * sigma_1p, 'r')
plot(Time, 2 * sigma_1p, 'b')
plot(Time, -2 * sigma_1p, 'b')
plot(Time, 3 * sigma_1p, 'g')
plot(Time, -3 * sigma_1p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in X-Coordinate (ft)')
title('X-Coordinate Error')
legend('Error in X', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

```

```
subplot(3, 2, 2)
plot(Time, error2p, 'k')
hold on
plot(Time, 1 * sigma_2p, 'r')
plot(Time, -1 * sigma_2p, 'r')
plot(Time, 2 * sigma_2p, 'b')
plot(Time, -2 * sigma_2p, 'b')
plot(Time, 3 * sigma_2p, 'g')
plot(Time, -3 * sigma_2p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{X}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{X}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{X}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )


subplot(3, 2, 3)
plot(Time, error3p, 'k')
hold on
plot(Time, 1 * sigma_3p, 'r')
plot(Time, -1 * sigma_3p, 'r')
plot(Time, 2 * sigma_3p, 'b')
plot(Time, -2 * sigma_3p, 'b')
plot(Time, 3 * sigma_3p, 'g')
plot(Time, -3 * sigma_3p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in Y-Coordinate (ft)')
title('Y-Coordinate Error')
legend('Error in Y', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
```

```
    sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 4)
plot(Time, error4p, 'k')
hold on
plot(Time, 1 * sigma_4p, 'r')
plot(Time, -1 * sigma_4p, 'r')
plot(Time, 2 * sigma_4p, 'b')
plot(Time, -2 * sigma_4p, 'b')
plot(Time, 3 * sigma_4p, 'g')
plot(Time, -3 * sigma_4p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{Y}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{Y}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{Y}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$
    ', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 5)
plot(Time, error5p, 'k')
hold on
plot(Time, 1 * sigma_5p, 'r')
plot(Time, -1 * sigma_5p, 'r')
plot(Time, 2 * sigma_5p, 'b')
plot(Time, -2 * sigma_5p, 'b')
plot(Time, 3 * sigma_5p, 'g')
plot(Time, -3 * sigma_5p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in Z-Coordinate (ft)')
title('Z-Coordinate Error')
legend('Error in Z', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
```

```
sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 6)
plot(Time, error6p, 'k')
hold on
plot(Time, 1 * sigma_6p, 'r')
plot(Time, -1 * sigma_6p, 'r')
plot(Time, 2 * sigma_6p, 'b')
plot(Time, -2 * sigma_6p, 'b')
plot(Time, 3 * sigma_6p, 'g')
plot(Time, -3 * sigma_6p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{Z}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{Z}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{Z}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )
toc
```

```
%% AAE590ET Project: Case 5
clear
clc
close all
rng('Default')
tic
% System Modeling
w = [1, 10^-1, 1]; % Process noise definition (standard deviations)
Q = diag([w(1)^2, w(2)^2, w(3)^2]); % (covariances)
G = [0, 0, 0; 1, 0, 0; 0, 0, 0; 0, 1, 0; 0, 0, 0; 0, 0, 1]; % Mapping Matrix
timespan = 0:0.1:500; % Defining timespan for noise
v = [1, 10, 10^-3, 10^-1]; % Measurement noise definition (standard deviations)
R = diag(v.^2);
```

```
tspan = 0:0.1:100;
x0 = [0, 180, 0, 90, 0, 10];

[noise] = getNoise(w, timespan); % Creating noise array
[Flight, Time] = getTrajectory(noise, G, tspan, x0); % System Dynamics Model


% Measurement Model
[range, range_rate, theta, phi] = getMeasurements(Flight, v);
[cart_traj] = getCartesian(range, theta, phi);

% Storage Setup
xkm_history = NaN(6, length(Time));
xkp_history = NaN(6, length(Time));
Pkm_history = NaN(6, 6, length(Time));
Pkp_history = NaN(6, 6, length(Time));


% Initial Conditions
x0_hat = [0, 171, 0, 99, 0, 11.96];
P0 = diag((x0-x0_hat).*(x0-x0_hat));
init_conditions = [x0_hat(:); P0(:)];
xkm_pr = x0_hat;
Pkm_pr = P0;
tkm = 0;
options = odeset('RelTol', 1e-10, 'AbsTol', 1e-10);

% Measurement Availability
settime = 0;
measurementstep = 1/4600;
zk = NaN(4, length(Time));

```

```
while settime <= 1
    instance = find(Time <= settime, 1, 'last');
    zk(:, instance) = [range(instance); range_rate(instance); theta(instance);
        phi(instance)];
    settime =  settime + measurementstep;

end

% Kalman Filtering
for i = 2:length(Time)

    % Priori Prediction
    tk = Time(i);
    span = [tkm, tk];
    init_conditions = [xkm_pr(:); Pkm_pr(:)];
    [t, propagate] = ode45(@(t, propagate) Propagate(t, propagate, G, Q), span
        , init_conditions); % This is right
    xkm = propagate(end, 1:6).';
    Pkm = reshape(propagate(end, 7:end), 6, 6);

    if isnan(zk(:, i)) == [0, 0, 0, 0];
    % Innovations
    H = getMeasurementJacobian(xkm);
    Wk = H*Pkm*H.' + R;
    Ck = Pkm*H.';

    % Gain
    Kk = Ck / Wk;

    % Update
    z_est = getEstimate(xkm)';
```

```
    Zk = zk(:, i);
    xkp = xkm + Kk * (zk(:, find(Time <= tk, 1, 'last')) - z_est);
    Pkp = Pkm - Ck*Kk.' - Kk*Ck.' + Kk*Wk*Kk.';
    Pkp = 0.5 * (Pkp + Pkp.');

    else
        xkp = xkm;
        Pkp = Pkm;

    end

    % Storage
    xkm_history(:, i) = xkm;
    xkp_history(:, i) = xkp;
    Pkm_history(:,:, i) = Pkm;
    Pkp_history(:,:,i) = Pkp;

    % Recursive
    xkm_pr = xkp;
    Pkm_pr = Pkp;
    tkm = tk;
end

figure
plot3(xkp_history(1, :), xkp_history(5, :), xkp_history(3, :), 'bo')
hold on
plot3(Flight(1, :), Flight(5, :), Flight(3, :), 'ro')
fill3([0, max(Flight(1, :))*1.1, max(Flight(1, :))*1.1, 0], [-max(Flight(5, :)
    ), -max(Flight(5, :)), max(Flight(5, :)), max(Flight(5, :))], [0, 0, 0,
    0], 'g')
grid on
xlabel('X Coordinates (ft)')
```

```
ylabel('Y Coordinates (ft)')
zlabel('Z Coordinates (ft)')
title('Kalman Filter Estimate of Ball Trajectory')
legend('Kalman Filter', 'True Trajectory')

figure
plot(Time, xkp_history(2, :))
hold on
plot(Time, Flight(2, :))

%% Sigma Bounds
sigma_1p = sqrt(squeeze(Pkp_history(1,1,:)));
sigma_1m = sqrt(squeeze(Pkm_history(1,1,:)));
sigma_2p = sqrt(squeeze(Pkp_history(2,2,:)));
sigma_2m = sqrt(squeeze(Pkm_history(2,2,:)));
sigma_3p = sqrt(squeeze(Pkp_history(3,3,:)));
sigma_3m = sqrt(squeeze(Pkm_history(3,3,:)));
sigma_4p = sqrt(squeeze(Pkp_history(4,4,:)));
sigma_4m = sqrt(squeeze(Pkm_history(4,4,:)));
sigma_5p = sqrt(squeeze(Pkp_history(5,5,:)));
sigma_5m = sqrt(squeeze(Pkm_history(5,5,:)));
sigma_6p = sqrt(squeeze(Pkp_history(6,6,:)));
sigma_6m = sqrt(squeeze(Pkm_history(6,6,:)));

%% Errors
error1p = Flight(1, :) - xkp_history(1,:);
error1m = Flight(1, :) - xkm_history(1,:);
error2p = Flight(2, :) - xkp_history(2,:);
error2m = Flight(2, :) - xkm_history(2,:);
error3p = Flight(3, :) - xkp_history(3,:);
error3m = Flight(3, :) - xkm_history(3,:);
error4p = Flight(4, :) - xkp_history(4,:);
```

```
error4m = Flight(4, :) - xkm_history(4,:);
error5p = Flight(5, :) - xkp_history(5,:);
error5m = Flight(5, :) - xkm_history(5,:);
error6p = Flight(6, :) - xkp_history(6,:);
error6m = Flight(6, :) - xkm_history(6,:);

%% Plot Estimation Error and Associated  3  Sigma Bounds
% Each State
figure
subplot(3, 2, 1)
plot(Time, error1p, 'k')
hold on
plot(Time, 1 * sigma_1p, 'r')
plot(Time, -1 * sigma_1p, 'r')
plot(Time, 2 * sigma_1p, 'b')
plot(Time, -2 * sigma_1p, 'b')
plot(Time, 3 * sigma_1p, 'g')
plot(Time, -3 * sigma_1p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in X-Coordinate (ft)')
title('X-Coordinate Error')
legend('Error in X', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
    sigma$', '$-3\sigma$',  'Interpreter', 'latex' )


subplot(3, 2, 2)
plot(Time, error2p, 'k')
hold on
plot(Time, 1 * sigma_2p, 'r')
plot(Time, -1 * sigma_2p, 'r')
plot(Time, 2 * sigma_2p, 'b')
```

```
plot(Time, -2 * sigma_2p, 'b')
plot(Time, 3 * sigma_2p, 'g')
plot(Time, -3 * sigma_2p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{X}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{X}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{X}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$
    ', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )


subplot(3, 2, 3)
plot(Time, error3p, 'k')
hold on
plot(Time, 1 * sigma_3p, 'r')
plot(Time, -1 * sigma_3p, 'r')
plot(Time, 2 * sigma_3p, 'b')
plot(Time, -2 * sigma_3p, 'b')
plot(Time, 3 * sigma_3p, 'g')
plot(Time, -3 * sigma_3p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in Y-Coordinate (ft)')
title('Y-Coordinate Error')
legend('Error in Y', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
    sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 4)
plot(Time, error4p, 'k')
hold on
plot(Time, 1 * sigma_4p, 'r')
plot(Time, -1 * sigma_4p, 'r')
```

```
plot(Time, 2 * sigma_4p, 'b')
plot(Time, -2 * sigma_4p, 'b')
plot(Time, 3 * sigma_4p, 'g')
plot(Time, -3 * sigma_4p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{Y}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{Y}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{Y}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$
    ', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 5)
plot(Time, error5p, 'k')
hold on
plot(Time, 1 * sigma_5p, 'r')
plot(Time, -1 * sigma_5p, 'r')
plot(Time, 2 * sigma_5p, 'b')
plot(Time, -2 * sigma_5p, 'b')
plot(Time, 3 * sigma_5p, 'g')
plot(Time, -3 * sigma_5p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in Z-Coordinate (ft)')
title('Z-Coordinate Error')
legend('Error in Z', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\
    sigma$', '$-3\sigma$',  'Interpreter', 'latex' )

subplot(3, 2, 6)
plot(Time, error6p, 'k')
hold on
plot(Time, 1 * sigma_6p, 'r')
plot(Time, -1 * sigma_6p, 'r')
```

```
plot(Time, 2 * sigma_6p, 'b')
plot(Time, -2 * sigma_6p, 'b')
plot(Time, 3 * sigma_6p, 'g')
plot(Time, -3 * sigma_6p, 'g')
grid on
xlabel('Time (s)')
ylabel('Error in $\dot{Z}$-Coordinate (ft)', 'Interpreter', 'latex')
title('$\dot{Z}$-Coordinate Error', 'Interpreter', 'latex')
legend('Error in $\dot{Z}$', '$\sigma$', '$-\sigma$', '$2\sigma$', '$-2\sigma$', '$3\sigma$', '$-3\sigma$',  'Interpreter', 'latex' )
toc
```

```
%% AAE590ET Project: Process Noise Determination
clear
clc
close all

% System Modeling
G = [0, 0, 0; 1, 0, 0; 0, 0, 0; 0, 1, 0; 0, 0, 0; 0, 0, 1]; % Mapping Matrix
timespan = 0:0.1:500; % Defining timespan for noise
v = [10^-2, 10^-2, 10^-3, 10^-3];
R = chol(diag(v), 'lower');
tspan = 0:0.1:100;
x0 = [0, 175, 0, 75, 0, 0];
rng('Default')
figure
for i = -5:3
    w = [10^i, 10^(i-1), 10^i]; % Process noise definition
    Q = diag([w(1), w(2), w(3)]);
    [noise] = getNoise(w, timespan); % Creating noise array
    [Flight, Time] = getTrajectory(noise, G, tspan, x0); % System Dynamics Model
    plot3(Flight(1, :), Flight(5, :), Flight(3, :))
```

```
    hold on
end
legend('i = -5', 'i = -4', 'i = -3', 'i = -2', 'i = -1', 'i = 0', 'i = 1', 'i = 2', 'i = 3')
xlabel('X Coordinate (ft)')
ylabel('Z Coordinate (ft)')
zlabel('Y Coordinate (ft)')
title('Determination of Process Noise i = [-5, 3]')
grid on

figure
for i = -5:1
    w = [10^i, 10^(i-1), 10^i]; % Process noise definition
    Q = diag([w(1), w(2), w(3)]);
    [noise] = getNoise(w, timespan); % Creating noise array
    [Flight, Time] = getTrajectory(noise, G, tspan, x0); % System Dynamics Model
    plot3(Flight(1, :), Flight(5, :), Flight(3, :))
    hold on
end
legend('i = -5', 'i = -4', 'i = -3', 'i = -2', 'i = -1', 'i = 0', 'i = 1')
xlabel('X Coordinate (ft)')
ylabel('Z Coordinate (ft)')
zlabel('Y Coordinate (ft)')
title('Determination of Process Noise i = [-5, 1]')
grid on
```

```
function [position, isterminal, direction] = hitground(t, x)
    position = x(3);
    isterminal = 1;
    direction = 0;
end
```

```matlab
%% AAE590ET Project: Get Trajectory
function [flight, t] = getTrajectory(noise, G, timespan, x0)
options = odeset('RelTol', 1e-10, 'AbsTol', 1e-10, 'Events', @(t, x) hitground
    (t, x));

[t, x] = ode45(@(t, x) flightg(t, x), [0, 500], x0, options);

flight = x';
function xprime = flightg(t, x)
    C_d = 0.23; % Between 0.24 and 0.7, typically
    r = 0.85; % Radius (in)
    rho = 0.0000442881929; % Atmospheric Density (lb/in^3)
    A = pi*(r/12)^2; % Area of a golf ball (in^2)
    D = ((1/2)*C_d*rho*A); % Drag Force (lb/in)
    s = 0.000005; % Magnus Coefficient (Assumed constant)
    m = 0.20235843; % Mass of golf ball (lbs)
    M = s/m; % Magnus coefficient divided by mass
    W_I = 110; % Spin in x-direction
    W_J = 110; % Spin in y-direction
    W_K = -110; % Spin in z-direction

    xprime(1) = x(2); % X'
    xprime(2)=-(D/m)*x(2)^2+M*(W_J*x(6)-W_K*x(4)); % X''
    xprime(3)=x(4); % Y'
    xprime(4)=-32.2-(D/m)*x(4)^2+M*(W_K*x(2)-W_I*x(6)); % Y''
    xprime(5)=x(6); % Z'
    xprime(6)=-(D/m)*x(6)^2+M*(W_I*x(4)-W_J*x(2)); % Z''

    xprime = xprime' + (G * noise(1:3, find(timespan <= t, 1, 'last')));

    xprime = xprime(:);
end
```

```
end
```

[14] [15] [16] [10]

```
function [noise] = getNoise(w, timespan)

    for i = 1:length(w)
        noise(i, :) = w(i) * randn(1, length(timespan));
    end
end
```

```
function [J] = getModelJacobian(x)
    C_d = 0.23; % Between 0.24 and 0.7, typically
    r = 0.85; % Radius (in)
    rho = 0.0000442881929; % Atmospheric Density (lb/in^3)
    A = pi*(r/12)^2; % Area of a golf ball (in^2)
    D = ((1/2)*C_d*rho*A); % Drag Force (lb/in)
    s = 0.000005; % Magnus Coefficient (Assumed constant)
    m = 0.20235843; % Mass of golf ball (lbs)
    M = s/m; % Magnus coefficient divided by mass
    W_I = 110; % Spin in x-direction
    W_J = 0; % Spin in y-direction
    W_K = -110; % Spin in z-direction

    J(1, :) = [0, 1, 0, 0, 0, 0];
    J(2, :) = [0, -2*D*x(2)/m, 0, -M*W_K, 0, M*W_J];
    J(3, :) = [0, 0, 0, 1, 0, 0];
    J(4, :) = [0, M*W_K, 0, -2*D*x(4)/m, 0, -M*W_I];
    J(5, :) = [0, 0, 0, 0, 0, 1];
    J(6, :) = [0, -M*W_J, 0, M*W_I, 0, -2*D*x(6)/m];
end
```

```
function [range, range_rate, theta, phi] = getMeasurements(T, v)

```

```
    for i = 1:length(T)
        traj = [T(1, i), T(3, i), T(5,i)];
        range(i) = (norm(traj)) + (randn * v(1));
        range_rate(i) = ((T(1, i)*T(2, i) + T(3, i)*T(4, i) + T(5, i)*T(6, i))
            / norm(traj)) + (randn * v(2));
        theta(i) = atan(T(5, i) / T(1, i)) + (randn * v(3));
        phi(i) = acos(T(3, i) / (norm(traj))) + (randn * v(4));
    end

    measurement = [range, range_rate, theta, phi];
end
```

```
function [J] = getMeasurementJacobian(x)
    C_d = 0.23; % Between 0.24 and 0.7, typically
    r = 0.85; % Radius (in)
    rho = 0.0000442881929; % Atmospheric Density (lb/in^3)
    A = pi*(r/12)^2; % Area of a golf ball (in^2)
    D = ((1/2)*C_d*rho*A); % Drag Force (lb/in)
    s = 0.000005; % Magnus Coefficient (Assumed constant)
    m = 0.20235843; % Mass of golf ball (lbs)
    M = s/m; % Magnus coefficient divided by mass
    W_I = 110; % Spin in x-direction
    W_J = 110; % Spin in y-direction
    W_K = -110; % Spin in z-direction

    T = [x(1), x(3), x(5)];
    J(1, 1) = x(1) / norm(T);
    J(1, 2) = 0;
    J(1, 3) = x(3) / norm(T);
    J(1, 4) = 0;
    J(1, 5) = x(5) / norm(T);
    J(1, 6) = 0;
    J(2, 1) = -(((x(6)*x(5) + x(4)*x(3))*x(1) - (x(5)^2 + x(3)^2)*x(2)) / norm
```

```
        (T)^3);
    J(2, 2) = x(1) / norm(T);
    J(2, 3) = -(((x(6)*x(5) + x(1)*x(2))*x(3) - (x(5)^2 + x(1)^2)*x(4)) / norm
        (T)^3);
    J(2, 4) = x(3) / norm(T);
    J(2, 5) = -(((x(3)*x(4) + x(1)*x(2))*x(5) - (x(3)^2 + x(5)^2)*x(6)) / norm
        (T)^3);
    J(2, 6) = x(5) / norm(T);
    J(3, 1) = -x(5) / (x(1)^2 + x(5)^2);
    J(3, 2) = 0;
    J(3, 3) = 0;
    J(3, 4) = 0;
    J(3, 5) = x(1) / (x(5)^2 + x(1)^2);
    J(3, 6) = 0;
    J(4, 1) = (x(1)*x(3)) / (norm(T)^3 * (1 - x(3)^2 / (norm(T)^2)));
    J(4, 2) = 0;
    J(4, 3) = - (x(1)^2 + x(5)^2) / (norm(T)^3 * (1 - x(3)^2 / (norm(T)^2)));
    J(4, 4) = 0;
    J(4, 5) = (x(5)*x(3)) / (norm(T)^3 * (1 - x(3)^2 / (norm(T)^2)));
    J(4, 6) = 0;

end
```

```
function [z_est] = getEstimate(x)
    T = [x(1), x(3), x(5)];
    z_est(1) = norm(T);
    z_est(2) = (x(1)*x(2) + x(3)*x(4) + x(5)*x(6)) / norm(T);
    z_est(3) = atan(x(5) / x(1));
    z_est(4) = acos(x(3) / norm(T));
end
```

```
function [cart_traj] = getCartesian(range, theta, phi)
    for i = 1:length(range)
```

```
        x(i) = range(i) * sin(phi(i)) * cos(theta(i));
        z(i) = range(i) * sin(phi(i)) * sin(theta(i));
        y(i)= range(i) * cos(phi(i));
    end
    cart_traj = [x', y', z'];
end
```