MAE 271B Project

1 Abstract

The goal of this project was to intercept a target with a missile using line-of sight measurements. Two models were used to develop the state dynamics in this project: the Gauss-Markov model and Random Telegraph model. The Continuous-time Kalman Filter was used to determine minimum variance estimates of the lateral position, velocity, and target acceleration for both models. These estimates were compared to their corresponding true values over a span of 10 seconds. A Monte Carlo simulation was run for 10,000 realizations in order to confirm the Kalman Filter algorithm was functional and the models used were approximately correct. A comparison of the root mean square error of the different states from the simulation is made with the corresponding filter values. The Kalman Filter performed similarly for simulations of both the Gauss-Markov model and Random Telegraph model.

2 Introduction

This project considers the relative state estimation of a missile intercept.

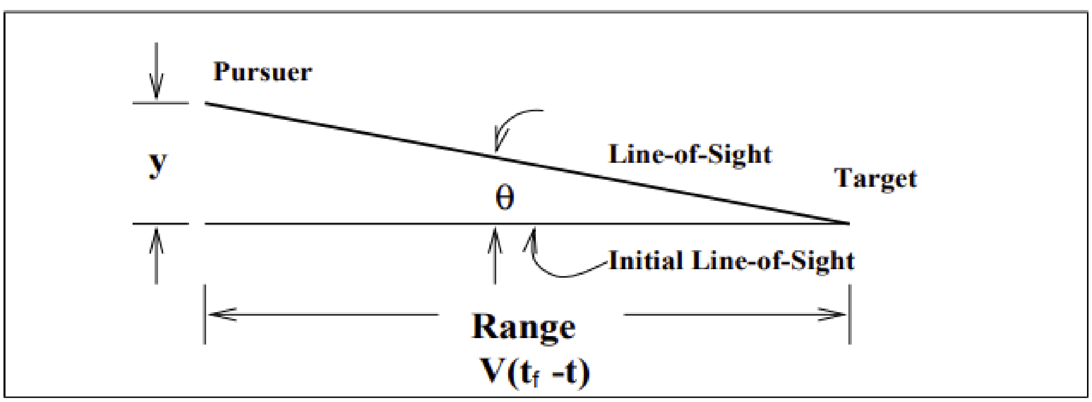


Figure 1: Missile Intercept Illustration

The dynamics of the problem are

$$
\begin{aligned}
\dot{y} &= v \\
\dot{v} &= a_P - a_T
\end{aligned} \tag{1}
$$

where a_P, the missile acceleration, is known and assumed here to be zero. The input, a_T is the target acceleration and is treated as a random forcing function with an exponential correlation,

$$
\begin{aligned}
E[a_T] &= 0 \\
E[a_T(t)a_T(s)] &= E[a_T^2]e^{\frac{-|t-s|}{\tau}}
\end{aligned}
$$

The scalar, τ, is the correlation time. The initial lateral position, $y(t_0)$ is zero by definition. The initial lateral velocity, $v(t_0)$, is random and assumed to be the result of launching error:

$$
\begin{aligned}
E[y(t_0)] = \;& 0 & E[v(t_0)] = \;& 0 & & \\
E[y(t_0)^2] = \;& 0 & E[y(t_0)v(t_0)] = \;& 0 & E[v(t_0)^2] &= [200 \text{ ft sec}^{-1}]^2
\end{aligned}
$$

The measurement, z, consists of a line-of-sight angle, θ. For $|\theta| \ll 1$

$$
\theta \approx \frac{y}{V_c(t_f - t)} \tag{2}
$$

It is also assumed that z is corrupted by fading and scintillation noise so that

$$
\begin{aligned}
z &= \theta + n \qquad (3) \\
E[n(t)] &= 0 \\
E[n(t)n(\tau)] &= V\delta(t-\tau) = \left[R_1 + \frac{R_2}{(t_f-t)^2}\right]\delta(t-\tau)
\end{aligned}
$$

The process noise spectral density, W, is

$$
W = GE[a_T^2]G^T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & E[a_T^2] \end{bmatrix}
$$

The considered parameters of the problem are

$$V_c = 300 \ \frac{\text{ft}}{\text{sec}}, \qquad t_f = \ 10 \text{ sec}, \qquad R_1 = 15 \times 10^{-6} \ \text{rad}^2\text{sec},$$
$$E[a_T^2] = [100\text{ft sec}^{-2}]^2, \qquad \tau = \ 2 \text{ sec}, \qquad R_2 = 1.67 \times 10^{-3} \ \text{rad}^2\text{sec}^3,$$

The initial covariance is

$$P(0) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & (200 \text{ ft sec}^{-1})^2 & 0 \\ 0 & 0 & (100 \text{ ft sec}^{-2})^2 \end{bmatrix} \tag{4}$$

which may be the parameters for a Falcon or Sparrow guided missile.

3 Theory and Algorithm

3.1 Gauss-Markov Model

A Gauss-Markov model was derived for the missile intercept states. We are assuming the acceleration of the pursuer is zero in this problem.

3.1.1 Derivation of Gauss-Markov State Space Equations

Utilizing the dynamics stated in Equation (1), the state space equation for the missile intercept problem is as follows:

$$\dot{x} = Fx(t) + Ba_P + Gw_{a_T}$$
$$\begin{Bmatrix} \dot{y} \\ \dot{v} \\ \dot{a_T} \end{Bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & -\frac{1}{\tau} \end{bmatrix}}_{F} \underbrace{\begin{Bmatrix} y \\ v \\ a_T \end{Bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}}_{B} a_P + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{G} w_{a_T} \tag{5}$$

with initial states as follows:

$$
\begin{aligned}
y(t_0) &= 0 \\
v(t_0) &= v_0 \sim N(0, P_0^v) \\
a_T(t_0) &= a_{T0} \sim N(0, P_0^{a_T}) \\
E[w_{a_T}] &= 0, E[w_{a_T}^2] = E[a_T^2]/dt
\end{aligned}
$$

The measurement, z, is a corrupted value of the line of sight angle θ. Using Equations (2) and (3), for θ and z, the state space model of the measurement can be summarized in the following Equation (6).

$$
z(t) = H(t)x(t) + n
$$

$$
z = \underbrace{\begin{bmatrix} \frac{1}{V_c(t_f - t)} & 0 & 0 \end{bmatrix}}_{H} \begin{Bmatrix} y \\ v \\ a_T \end{Bmatrix} + n \tag{6}
$$

with initial states as follows:

$$
E[n] \quad = \quad 0, E[n^2] = V/dt
$$

3.2 Random Telegraph Signal Model

The Gauss-Markov model is an approximation to the more realistic, Random Telegraph Signal model. The objective of using the random telegraph signal as well is to ensure that the Kalman filter is implemented correctly.

3.2.1 Generation of Random Telegraph Signal

In the random telegraph signal model, a_T changes sign at random times given by a Poisson probability. We assume that $a_T(0) = \pm a_T$ with probability 0.5 and a_T changes polarity at Poisson times. The probability of k sign changes in a time interval of length T, $P(k(T))$, is $P(k(T)) = \frac{(\lambda^T)^k e^{-\lambda^T}}{k!}$ where λ is the rate.

The method for generating random switching times is developed by letting $T = t_{n+1} - t_n$ be the time between two switch times.

The probability that the switch occurred after t_{n+1} is

$$P(T' > T | t = t_n) = 1 - P(T' \leq T | t = t_n)$$

The probability that no switch occurred in T, but occurred after t_{n+1}, is

$$P(T' > T | t = t_n) = P(\text{\# of sign changes in T is zero}) = e^{-\lambda T}$$

Then,

$$P(T' \leq T | t = t_n) = 1 - e^{-\lambda T}$$

which is the probability that at least one change occurred. To produce the random time t_{n+1}, set $P(T' \leq T | t = t_n)$ equal to U, the output of [0,1] from a uniform density function. Then,

$$\begin{aligned} 1 - e^{-\lambda T} = U \quad &\rightarrow \quad e^{-\lambda T} = 1 - U \\ &\rightarrow \quad -\lambda T = \ln(1 - U) \\ &\rightarrow \quad T = \frac{-1}{\lambda}\ln(1 - U) \\ &\rightarrow \quad t_{n+1} = t_n - \frac{1}{\lambda}\ln(1 - U) \end{aligned}$$

Because $1 - U$ is also a uniform density function, the random switching time can be defined as Equation (7).

$$t_{n+1} = t_n - \frac{1}{\lambda}\ln(U) \tag{7}$$

An example of the random signal produced at random switching times and for random initial state $a_T(0)$ is shown in Figure 2.

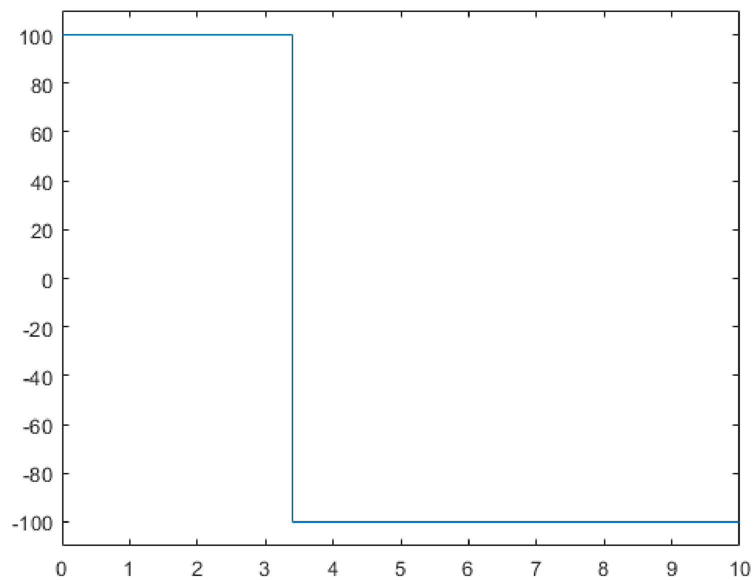


Figure 2: Random Telegraph Target Acceleration

3.2.2 Derivation of Telegraph State Space Equations

Because the true state of a_T is now given by the random telegraph signal, the true states of position and velocity are now propagated by a two-state model given in Equation (8).

$$\dot{x} = F_t x(t) + G_t a_T$$
$$\begin{Bmatrix} \dot{y} \\ \dot{v} \end{Bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}}_{F_t} \begin{Bmatrix} y \\ v \end{Bmatrix} + \underbrace{\begin{bmatrix} 0 \\ -1 \end{bmatrix}}_{G_t} a_T \tag{8}$$

The random telegraph signal now directly propagates the third state of vector x where $x = [y, v, a_T]'$.

$$x(3, t_{i+1}) = a_T(t_{i+1})$$

The measurements are defined similarly as in the Gauss-Markov model given by Equation (9).

$$z(t) = H(t)x(t) + n$$

$$z = \underbrace{\begin{bmatrix} \frac{1}{V_c(t_f - t)} & 0 & 0 \end{bmatrix}}_{H} \begin{Bmatrix} y \\ v \\ a_T \end{Bmatrix} + n \tag{9}$$

3.3 Continuous-Time Kalman Filter

The continuous-time Kalman filter can be used on continuous-time systems represented by Itô stochastic differentials:

$$dx(t) = F(t)x(t)dt + G(t)d\beta_t \tag{10}$$

$$dz(t) = H(t)x(t)dt + dn \tag{11}$$

For the missile intercept problem, matrices F and G are time invariant and Brownian Motion process, $d\beta_t = w_{a_T}dt$.

3.3.1 Filter Algorithm

The continuous Kalman Filter has no separation between the propagation and measurement update stages. They occur simultaneously by Equation (12).

state propagation and update:

$$\begin{aligned} d\hat{x}(t) &= F(t)\hat{x}(t)dt + K(t)[dz(t) - H(t)\hat{x}(t)dt] \\ K(t) &= P(t)H(t)^T V(t)^{-1} \end{aligned} \tag{12}$$

covariance update (Riccati Equation):

$$\begin{aligned} \dot{P}(t) &= F(t)P(t) + P(t)F(t)^T - P(t)H(t)^T V(t)^{-1} H(t)P(t) + G(t)W(t)G(t)^T \\ P(0) &= P_0 \end{aligned}$$

For the missile intercept problem the Riccati equation becomes

$$\dot{P}(t) = FP(t) + P(t)F^T - \frac{1}{V_c^2 R_1 (t_f - t)^2 + V_c^2 R_2} P\bar{H}^T \bar{H} P + W \tag{13}$$

with $\bar{H} = [1, 0, 0]$

where the Kalman gains are the following

$$\begin{aligned} K_1 &= \frac{P_{11}}{V_c R_1 (t_f - t) + \frac{V_c R_2}{t_f - t}} \\ K_2 &= \frac{P_{12}}{V_c R_1 (t_f - t) + \frac{V_c R_2}{t_f - t}} \\ K_3 &= \frac{P_{13}}{V_c R_1 (t_f - t) + \frac{V_c R_2}{t_f - t}} \end{aligned}$$

scalars P_{ij} are the (i, j) elements of the error covariance matrix.

residuals:

$$dr(t) = dz(t) - H(t)\hat{x}(t)dt \tag{14}$$

3.4 Validation Analysis

One validation technique for the implemented Kalman Filter is to prove the residuals or “innovations” should be an independent sequence.

$$E[r_k r_j^T] = 0 \quad \forall j < k \tag{15}$$

3.4.1 Validation Algorithm

To validate the Kalman filter, a Monte Carlo simulation was used to compute an ensemble of realizations of the state and state estimates. Over the ensemble of realizations, the actual error and error variance can be calculated by averaging the errors of each realization. The independence of residuals can also be simulated over the ensemble.

It is expected that the ensemble average of the actual error for realization l, $e^l(t_i)$, is approximately zero ($e_{ave}(t_i) \approx 0$) for all $t_i \in [0, 30]$).

$$e^{ave}(t_i) = \frac{1}{N_{ave}} \sum_{l=1}^{N_{ave}} e^l(t_i) \tag{16}$$

The ensemble average produces the actual error variance P^{ave}. The matrix $P^{ave}(t_i)$ should be close to $P(t_i)$ computed in the Kalman filter algorithm.

$$P^{ave}(t_i) = \frac{1}{N_{ave} - 1} \sum_{l=1}^{N_{ave}} [e^l(t_i) - e^{ave}(t_i)][e^l(t_i) - e^{ave}(t_i)]^T \tag{17}$$

The residuals should be an independent sequence and this orthogonality property can be proven with Equation (18).

independence of residuals check:

$$\frac{1}{N_{ave}} \sum_{l=1}^{N_{ave}} r^l(t_i) r^l(t_m)^T \approx 0 \quad \forall \; t_m < t_i \tag{18}$$

4 Results and Performance

4.1 State Estimates

The estimated position, velocity and target acceleration were calculated using the Kalman filter algorithm (Section 3.3.1). The estimation errors, $x(t_i) - \hat{x}(t_i)$, for the Gauss-Markov model are plotted in Figure 3 and for the Random Telegraph model in Figure 4. The one sigma bounds are plotted against the error in order to visualize the decreasing filter error variance as time increases.

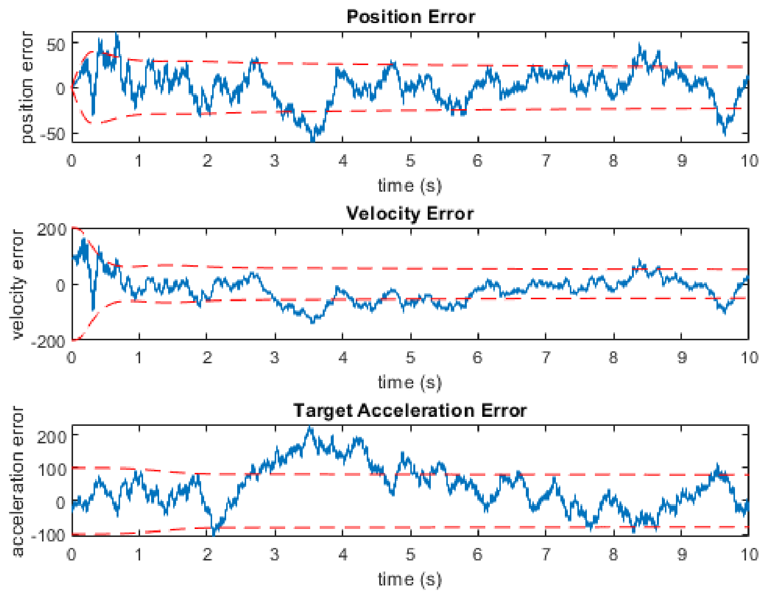


Figure 3: G-M Estimation Error $x(t_i) - \hat{x}(t_i)$

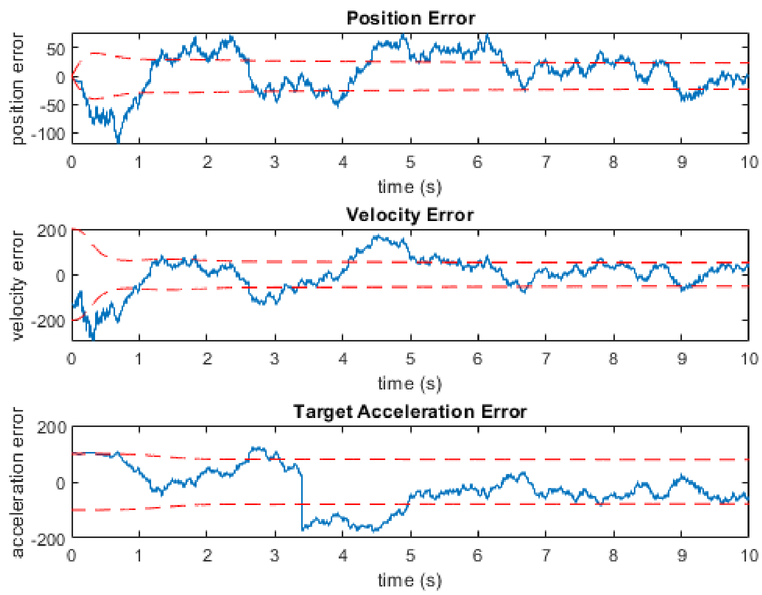


Figure 4: Telegraph Estimation Error $x(t_i) - \hat{x}(t_i)$

In addition, the position, velocity, and target acceleration estimates were plotted with their corresponding true values (See Figure 5 for Gauss-Markov and Figure 6 for Telegraph). Section 3 provides the truth model propagation equations for $y(t_i)$, $v(t_i)$, and $a_T(t_i)$ for both the Gauss-Markov model and Telegraph model. As expected, the estimates track the true value over time.

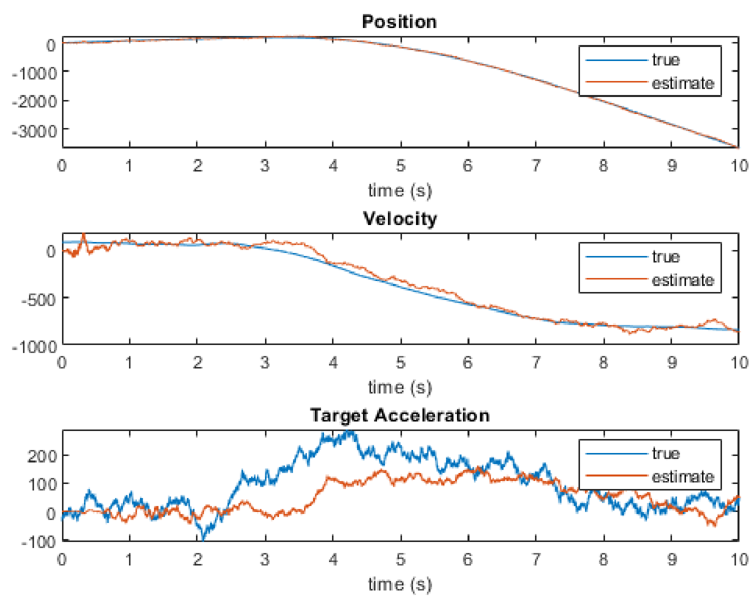


Figure 5: G-M State Estimate, $\hat{x}(t_i)$

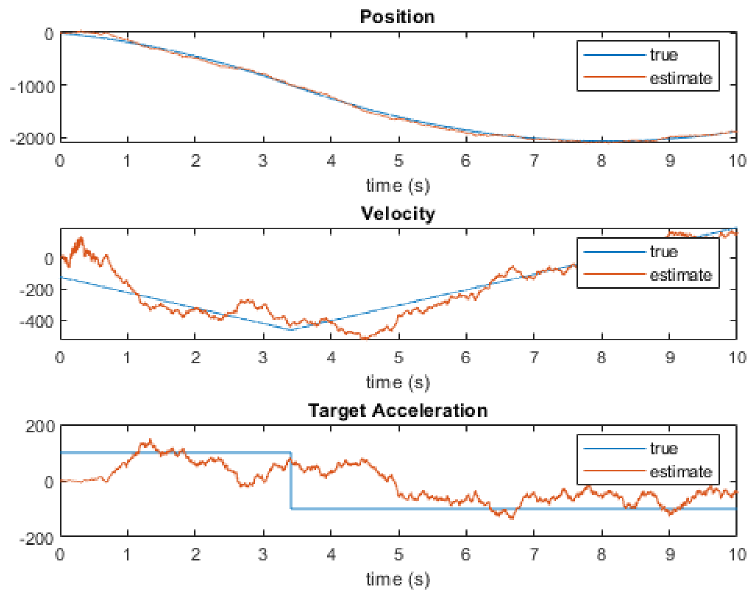


Figure 6: Telegraph State Estimate, $\hat{x}(t_i)$

4.2 Error Covariance and Kalman Gain Propagation

Both the Gauss-Markov model and Telegraph model produced the same Figures 7 and 8 for filter gain history as well as standard deviation of the state error. These results will be used to compare to the actual standard deviation of state error from a Monte Carlo Simulation.

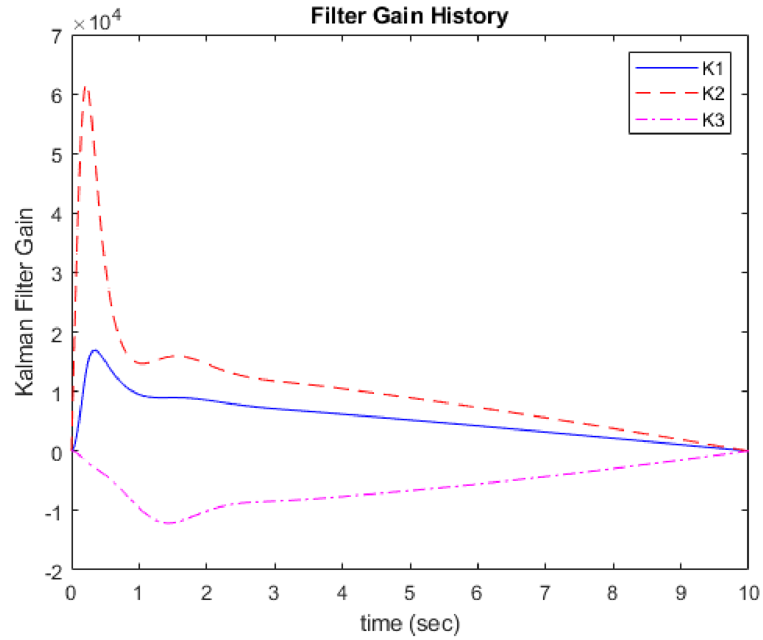


Figure 7: Evolution of Kalman Gain

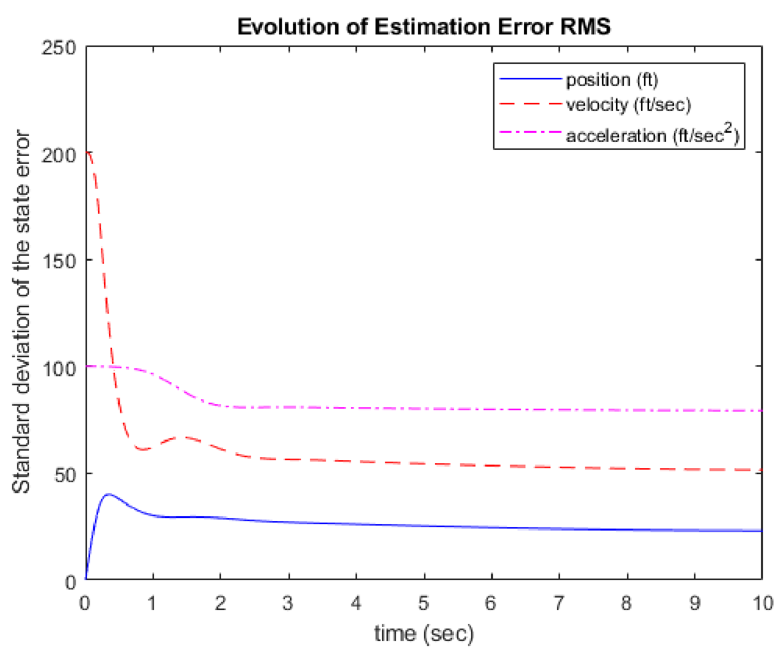
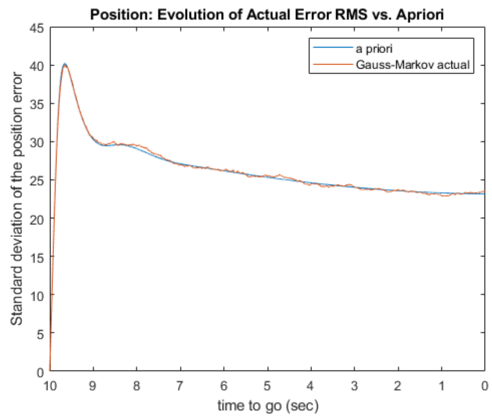


Figure 8: Evolution of Estimation Error RMS

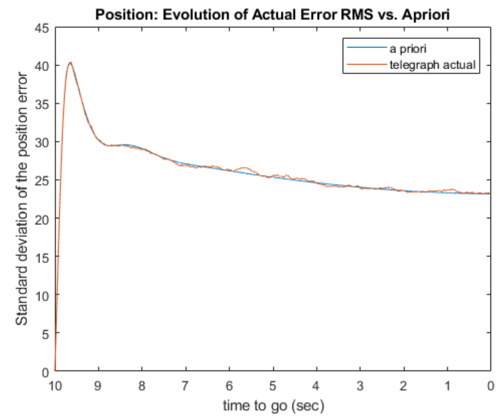
4.3 Filter Validation and Simulation

The derived Kalman Filter was validated by running a Monte Carlo Simulation with $N = 10,000$ realizations for both the Gauss-Markov and Telegraph model. Residual correlation was analyzed over the ensemble and a comparison of the state error standard deviation of the filter versus the ensemble average was made.

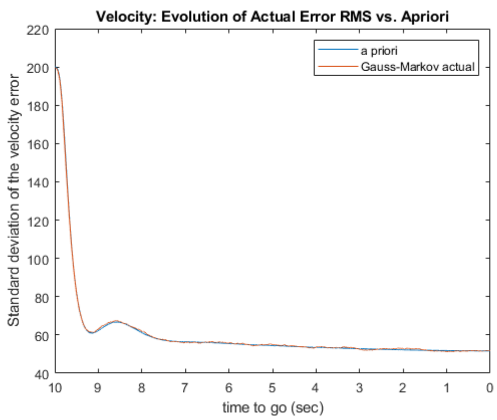
First, the filter was validated by comparing the simulated error variance, P^{ave} found with Equation (17), with the filter error variance derived from one realization, P. Figure 9 shows the comparison between the standard deviation of the state errors of both the models ($\sqrt{P^{ave}}$ diagonals plotted against $\sqrt{P}$ diagonals). The actual root-mean square error for the lateral position and velocity of the two models are close to the filter results. The actual root mean square error for the target acceleration using the Gauss-Markov model is initially noisier than that of the Telegraph model but otherwise yield similar results and track the *a priori* result.



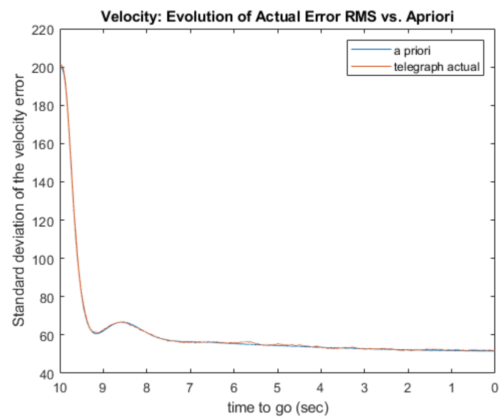
(a) G-M Lateral Position STD



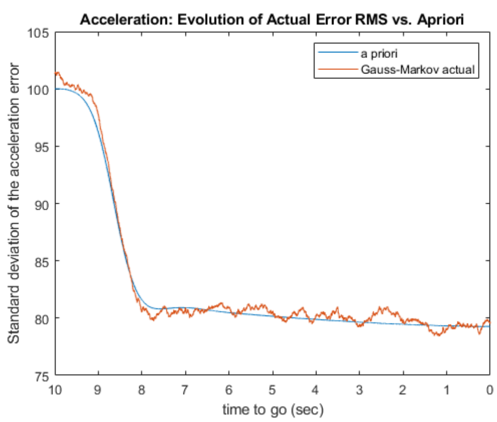
(b) Telegraph Lateral Position STD



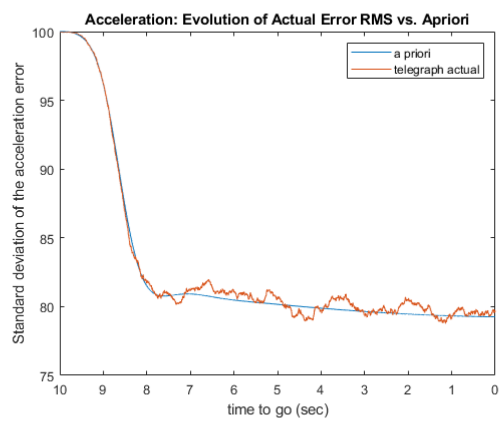
(c) G-M Lateral Velocity STD



(d) Telegraph Lateral Velocity STD



(e) G-M Target Acceleration STD



(f) Telegraph Target Acceleration STD

Figure 9: Standard Deviation of Filter vs. Ensemble Average

The independence of the residuals was checked using Equation (18). This was calculated for the residuals mid-simulation ($t_i = 5s$) and time step prior ($t_m = 4.999s$). The result is approximately zero, indicating the residuals form an independent sequence.

```
g-m model:
ensemble average for correlation of the residuals:
   7.6102e-04


telegraph model:
ensemble average for correlation of the residuals:
    0.0011
```

5 Conclusion

To conclude, the derived Kalman Filter was successful in determining the minimum variance estimates of the missile's lateral position, velocity and the target acceleration. The Kalman Filter behaved similarly for both the Gauss-Markov model and the Random Telegraph Signal model. The estimation error of each parameter generally stayed within the one sigma bounds with the exception of some short time periods. Although the Gauss-Markov model is linear while the Telegraph model is nonlinear, this project shows that the Kalman filter is the best linear filter even for a nonlinear model. The theoretical properties of the system also were validated using a Monte Carlo Simulation; namely checking the simulation error variance was approximately the filter error variance, and proving the independence of residuals. Proving these theoretical properties through simulation for both models confirmed a functional Kalman Filter algorithm.

A G-M Monte Carlo Simulation

```
%Helene Levy
%MAE 271B Project
%Monte Carlo Simulation
clc; clear; close all;

%time step choices
dt = 0.001;
tf = 10;
t = 0:dt:tf;

%preallocations
e_sum = zeros(3,length(t));
e_ave = zeros(3,length(t));
r_sum = zeros(1,1);

P_ave = zeros(3,3,length(t));
P_sum = zeros(3,3,length(t));
% ortho_ave = zeros(3,3,length(t));
% ortho_sum = zeros(3,3,length(t));

%iterating through N times
N = 10000;
for j = 1:N
    [X_hat,e_hat,P,r] = ct_kalman_filter(dt);

    %ensemble average of error
    e_sum = e_sum + e_hat;
    e_ave = 1/j*e_sum;

    for k = 1:length(e_hat)
```

```
        %actual error variance calculation
        P_sum(:,:,k) = P_sum(:,:,k)+(e_hat(:,k)-e_ave(:,k))...
                       *(e_hat(:,k)-e_ave(:,k))';
        P_ave(:,:,k) = 1/(j-1)*P_sum(:,:,k);

%         ortho_sum(:,:,k) = ... ortho_sum(:,:,k)+(e_hat(:,k)-e_ave(:,k))*X_hat(:,k)';
%         ortho_ave(:,:,k) = 1/j*ortho_sum(:,:,k);
    end

    %ensemble average for correlation of residuals at end time
    r_sum = r_sum + r(end/2)*(r(end/2-1))';
    r_ave = 1/j*r_sum;
end

%showing residuals uncorrelated
disp('ensemble average for correlation of the residuals:');
disp(r_ave);

%apriori standard deviation and actual standard deviation
std_true = sqrt([squeeze(P(1,1,:)), squeeze(P(2,2,:)), squeeze(P(3,3,:))]);
std_ave = sqrt([squeeze(P_ave(1,1,:)), squeeze(P_ave(2,2,:)),...
                squeeze(P_ave(3,3,:))]);

%plotting std comparison
figure;
plot(tf-t,std_true(:,1)); hold on;
plot(tf-t,std_ave(:,1));
set ( gca, 'xdir', 'reverse' );
xlabel('time to go (sec)');
ylabel('Standard deviation of the position error');
title('Position: Evolution of Actual Error RMS vs. Apriori')
legend('a priori','Gauss-Markov actual')
```

```
figure;
plot(tf-t,std_true(:,2)); hold on;
plot(tf-t,std_ave(:,2));
set ( gca, 'xdir', 'reverse' );
xlabel('time to go (sec)');
ylabel('Standard deviation of the velocity error');
title('Velocity: Evolution of Actual Error RMS vs. Apriori')
legend('a priori','Gauss-Markov actual')

figure;
plot(tf-t,std_true(:,3)); hold on;
plot(tf-t,std_ave(:,3));
set ( gca, 'xdir', 'reverse' );
xlabel('time to go (sec)');
ylabel('Standard deviation of the acceleration error');
title('Acceleration: Evolution of Actual Error RMS vs. Apriori')
legend('a priori','Gauss-Markov actual')

% cross terms in covariance matrix plotting
% figure;
% for i = 1:size(P_ave,1)
%     for k = 1:size(P_ave,2)
%         dP = squeeze(P_ave(i,k,:))-squeeze(P(i,k,:));
%         plot(t,dP); hold on;
%     end
% end
%
% legend('P(1,1)','P(1,2)','P(1,3)','P(2,1)','P(2,2)','P(2,3)','P(3,1)',...
%         'P(3,2)','P(3,3)','Location','southeast');
% title('Simulated - Filter Error Variance, P_{ave} - P');
% xlabel('time (s)');
% ylabel('P_{ave} - P');
```

B G-M Kalman Filter Function

```
function [Xhat,Ehat,P,r] = ct_kalman_filter(dt)
% function for running continuous time kalman filter

% Given parameters and statistics

% time parameters
tf = 10; % sec
tau = 2; % sec

R1 = 15*10^(-6); %rad^2/sec
R2 = 1.67*10^(-3); %rad^2/sec^3
Vc = 300; %ft/sec

% target acceleration
m_at = 0;
var_at = 100^2; % (ft/sec^2)^2

% lateral position
m_y = 0;
var_y = 0;

% lateral velocity
m_v = 0;
var_v = 200^2; % (ft/sec)^2

%fading and scintillation noise
m_n = 0;
V = @(t) R1 + R2/(tf-t)^2 ;

% process noise spectral density
```

```
W = [ 0 0 0; 0 0 0; 0 0 var_at];

% initial covariance
P0 = [var_y 0 0; 0 var_v 0; 0 0 var_at];

% State Space Matrices
% xdot = Fc + Bap + Gw_at
% x = [y v at]'
F = [0 1 0; 0 0 -1; 0 0 -1/tau];
B = [0; 1; 0];
G = [0; 0; 1];

H = @(t) [1/(Vc*(tf - t)) 0 0];
Hbar = [1 0 0];

%K and P calculations using ode45
tspan = 0:dt:tf;
[t,P] = ode45(@Pdot, tspan , P0(:));
P = reshape(P.',3,3,[]);

%calculating Kalman gains
K1 = squeeze(P(1,1,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));
K2 = squeeze(P(1,2,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));
K3 = squeeze(P(1,3,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));

%initial states
y_0 = m_y + sqrt(var_y)*randn(1,1); %(0)
v_0 = m_v + sqrt(var_v)*randn(1,1);
at_0 = m_at + sqrt(var_at)*randn(1,1);

% noise generation
w_at = m_at + sqrt(var_at/dt)*randn(1,length(t)-1);
n = @(t) m_n + sqrt(V(t)/dt)*randn(1);
```

```

%preallocating matrices
Xhat = zeros(3,length(t));
X = zeros(3,length(t));
z = zeros(1,length(t)-1);
r = zeros(1,length(t)-1);

X0 = [y_0 v_0 at_0]';

%initial conditions of state
X(:,1) = X0;
a_p = 0;
for i = 1: length(t)-1
    %true states
    dX = (F*X(:,i)+B*a_p+G*w_at(i))*dt;
    X(:,i+1) = X(:,i) + dX;

    %measurement
    z(i) = H(t(i))*X(:,i)+n(t(i));

    %residuals
    r(i) = z(i)-H(t(i))*Xhat(:,i);

    %estimates
    K = [K1(i) K2(i) K3(i)]';
    dXhat = F*Xhat(:,i)*dt + K*(z(i)-H(t(i))*Xhat(:,i))*dt;
    Xhat(:,i+1) = Xhat(:,i) + dXhat;

end
Ehat = X - Xhat;
end
```

C G-M Kalman Filter (One Realization)

```
% MAE 271B Project
% Helene Levy
clc; clear; close all;

%% Given parameters and statistics

% time parameters
tf = 10; % sec
tau = 2; % sec

R1 = 15*10^(-6); %rad^2/sec
R2 = 1.67*10^(-3); %rad^2/sec^3
Vc = 300; %ft/sec

% target acceleration
m_at = 0;
var_at = 100^2; % (ft/sec^2)^2
corr_at_as = @(t,s) var_at*exp(-(t-s)/tau);

% lateral position
m_y = 0;
var_y = 0;

% lateral velocity
m_v = 0;
var_v = 200^2; % (ft/sec)^2
corr_yv = 0;

%fading and scintillation noise
```

```
m_n = 0;
V = @(t) R1 + R2/(tf-t)^2 ;
corr_nt_ntau = @(t) (R1 + R2/(tf-t)^2)*dirac(t-tau);

% process noise spectral density
W = [ 0 0 0; 0 0 0; 0 0 var_at];

% initial covariance
P0 = [var_y 0 0; 0 var_v 0; 0 0 var_at];


%% State Space Matrices
% xdot = Fc + Bap + Gw_at
% x = [y v at]'
F = [0 1 0; 0 0 -1; 0 0 -1/tau];
B = [0; 1; 0];
G = [0; 0; 1];

H = @(t) [1/(Vc*(tf - t)) 0 0];
Hbar = [1 0 0];

%% P, K calculation and Plotting Figures 9.6 and 9.7
dt = 0.0001;
tspan = 0:dt:tf;
[t,P] = ode45(@Pdot, tspan , P0(:));
P = reshape(P.',3,3,[]);

%calculating Kalman gains
K1 = squeeze(P(1,1,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));
K2 = squeeze(P(1,2,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));
K3 = squeeze(P(1,3,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));

%plotting Kalman gains vs. time
```

```
figure;
plot(t,K1,'b-'); hold on;
plot(t,K2,'r--'); hold on;
plot(t,K3,'m-.');

legend('K1','K2','K3');
xlabel('time (sec)');
ylabel('Kalman Filter Gain');
title('Filter Gain History');

figure;
rms_y = sqrt(squeeze(P(1,1,:)));
rms_v = sqrt(squeeze(P(2,2,:)));
rms_at = sqrt(squeeze(P(3,3,:)));
plot(t,rms_y,'b-'); hold on;
plot(t,rms_v,'r--'); hold on;
plot(t,rms_at,'m-.');

legend({'position (ft)','velocity (ft/sec)','acceleration (ft/sec^2)'});
xlabel('time (sec)');
ylabel('Standard deviation of the state error');
title('Evolution of Estimation Error RMS');

%% Kalman Filter
%initial states
y_0 = m_y + sqrt(var_y)*randn(1,1); %(0)
v_0 = m_v + sqrt(var_v)*randn(1,1);
at_0 = m_at + sqrt(var_at)*randn(1,1);

% noise generation
w_at = m_at + sqrt(var_at/dt)*randn(1,length(t)-1);
n = @(t) m_n + sqrt(V(t)/dt)*randn(1);

```

```
X0 = [y_0 v_0 at_0]';

Xhat = zeros(3,length(t));

X = zeros(3,length(t));
X(:,1) = X0;
a_p = 0;

for i = 1: length(t)-1
    %true states
    dX = (F*X(:,i)+B*a_p+G*w_at(i))*dt;
    X(:,i+1) = X(:,i) + dX;

    %measurement
    z = H(t(i))*X(:,i)+n(t(i));

    %estimates
    K = [K1(i) K2(i) K3(i)]';
    dXhat = F*Xhat(:,i)*dt + K*(z-H(t(i))*Xhat(:,i))*dt;
    Xhat(:,i+1) = Xhat(:,i) + dXhat;
end
Ehat = X - Xhat;
sig= sqrt([squeeze(P(1,1,:)), squeeze(P(2,2,:)), squeeze(P(3,3,:))]);

%plotting states
figure;
subplot(311)
plot(t,X(1,:)); hold on;
plot(t,Xhat(1,:))
legend('true', 'estimate');
title('Position');
xlabel('time (s)');

```

```
subplot(312)
plot(t,X(2,:)); hold on;
plot(t,Xhat(2,:))
legend('true', 'estimate');
title('Velocity');
xlabel('time (s)');

subplot(313)
plot(t,X(3,:)); hold on;
plot(t,Xhat(3,:))
legend('true', 'estimate');
title('Target Acceleration');
xlabel('time (s)');

%plotting state errors

figure;
subplot(311)
stairs(t,Ehat(1,:));hold on;
plot(t,sig(:,1),'r--',t,-sig(:,1),'r--');
title('Position Error');
xlabel('time (s)');
ylabel('position error');

subplot(312)
stairs(t,Ehat(2,:)); hold on;
plot(t,sig(:,2),'r--',t,-sig(:,2),'r--');
title('Velocity Error');
xlabel('time (s)');
ylabel('velocity error');

subplot(313)
stairs(t,Ehat(3,:)); hold on;
```

```
plot(t,sig(:,3),'r--',t,-sig(:,3),'r--');
title('Target Acceleration Error');
xlabel('time (s)');
ylabel('acceleration error');
```

D Telegraph Monte Carlo Simulation

```
%Helene Levy
%MAE 271B Project
%Monte Carlo Simulation
clc; clear; close all;

%time step choices
dt = 0.001;
tf = 10;
t = 0:dt:tf;

%preallocations
e_sum = zeros(3,length(t));
e_ave = zeros(3,length(t));
r_sum = zeros(1,1);

P_ave = zeros(3,3,length(t));
P_sum = zeros(3,3,length(t));

%iterating through N times
N = 10000;
for j = 1:N
    [X_hat,e_hat,P,r] = tele_kalman_filt(dt);

```

```
    %ensemble average of error
    e_sum = e_sum + e_hat;
    e_ave = 1/j*e_sum;

    for k = 1:length(e_hat)
        %actual error variance calculation
        P_sum(:,:,k) = P_sum(:,:,k)+(e_hat(:,k)-e_ave(:,k))...
                       *(e_hat(:,k)-e_ave(:,k))';
        P_ave(:,:,k) = 1/(j-1)*P_sum(:,:,k);

    end

    %ensemble average for correlation of residuals at end time
    r_sum = r_sum + r(end/2)*(r(end/2-1))';
    r_ave = 1/j*r_sum;
end

%showing residuals uncorrelated
disp('ensemble average for correlation of the residuals:');
disp(r_ave);

%apriori standard deviation and actual standard deviation
std_true = sqrt([squeeze(P(1,1,:)), squeeze(P(2,2,:)), squeeze(P(3,3,:))]);
std_ave = sqrt([squeeze(P_ave(1,1,:)), squeeze(P_ave(2,2,:)),...
                squeeze(P_ave(3,3,:))]);

%plotting std comparison
figure;
plot(tf-t,std_true(:,1)); hold on;
plot(tf-t,std_ave(:,1));
set ( gca, 'xdir', 'reverse' );
xlabel('time to go (sec)');
ylabel('Standard deviation of the position error');
```

```
title('Position: Evolution of Actual Error RMS vs. Apriori')
legend('a priori','telegraph actual')

figure;
plot(tf-t,std_true(:,2)); hold on;
plot(tf-t,std_ave(:,2));
set ( gca, 'xdir', 'reverse' );
xlabel('time to go (sec)');
ylabel('Standard deviation of the velocity error');
title('Velocity: Evolution of Actual Error RMS vs. Apriori')
legend('a priori','telegraph actual')

figure;
plot(tf-t,std_true(:,3)); hold on;
plot(tf-t,std_ave(:,3));
set ( gca, 'xdir', 'reverse' );
xlabel('time to go (sec)');
ylabel('Standard deviation of the acceleration error');
title('Acceleration: Evolution of Actual Error RMS vs. Apriori')
legend('a priori','telegraph actual')
```

E Telegraph Kalman Filter Function

```
function [Xhat,Ehat,P,r] = tele_kalman_filt(dt)
% Given Parameters and Statistics
% time parameters
tf = 10; % sec
tau = 2; % sec
t = 0:dt:tf;

% lateral position
m_y = 0;
var_y = 0;

% lateral velocity
m_v = 0;
var_v = 200^2; % (ft/sec)^2
corr_yv = 0;

% target acceleration
m_at = 0;
var_at = 100^2; % (ft/sec^2)^2

% target acceleration
m_aTbar = 0;
aT = 100; %ft/sec^2
RaT = @(t,s) aT^2*exp(-2*lambda*abs(t-s));

%relative velocity
R1 = 15*10^(-6); %rad^2/sec
R2 = 1.67*10^(-3); %rad^2/sec^3
Vc = 300; %ft/sec

```

```
% process noise spectral density
W = [ 0 0 0; 0 0 0; 0 0 var_at];

%fading and scintillation noise
m_n = 0;
V = @(t) R1 + R2/(tf-t)^2 ;

% initial covariance
P0 = [var_y 0 0; 0 var_v 0; 0 0 var_at];

%Telegraph Process Generation
%poisson variable param
lambda = 0.25; %/sec

a =[];

%initial aT
A = rand;
if A ≤ 0.5
    a(1) = aT;
else
    a(1) = -aT;
end

%generating random switching times for telegraph signal
time = [];
time(1) = 0; i = 1;
t_np1 = 0;
%only want times in our simulation <10
while t_np1 < 10
    %uniform random variable
    U = rand;
    %next random switching time
```

```
    t_np1 = time(i) - 1/lambda*log(U);
    if t_np1 < 10
        time(i+1) = t_np1;
        a(i+1) = -a(i);
    end
    i = i+1;
end

%rounding the random time to accuracy of dt
time = round(time,-log10(dt));
t = round(t,-log10(dt));
[¬,loc] = ismember(time,t);

%converting a to timescale
at = zeros(1,length(t));
for j = 2:length(loc)
    at(loc(j-1):loc(j)) = a(j-1);
end
at(loc(end):end) = a(end);

% %visualizing telegraph signal
% stairs(t,at)

% Filter
% State space matrices
%two state telegraph
Ft = [0 1; 0 0];
Gt = [0; -1];

%original three state
F = [0 1 0; 0 0 -1; 0 0 -1/tau];
B = [0; 1; 0];
G = [0; 0; 1];
```

```
H = @(t) [1/(Vc*(tf-t)) 0 0];
Hbar = [1 0 0];

%initial states
y_0 = m_y + sqrt(var_y)*randn(1,1); %(0)
v_0 = m_v + sqrt(var_v)*randn(1,1);

% noise generation
n = @(t) m_n + sqrt(V(t)/dt)*randn(1);

X0 = [y_0 v_0 at(1)]';

%preallocation matrices
Xhat = zeros(3,length(t));
X = zeros(3,length(t));
z = zeros(1,length(t)-1);
r = zeros(1,length(t)-1);

P = zeros(3,3,length(t));

K1 = zeros(1,length(t));
K2 = zeros(1,length(t));
K3 = zeros(1,length(t));

%intitial values
X(:,1) = X0;
P(:,:,1) = P0;
K1(1) = P(1,1,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));
K2(1) = P(1,2,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));
K3(1) = P(1,3,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));

for i = 1: length(t)-1
    %true states
```

```
    dX = (Ft*X(1:2,i)+Gt*at(i))*dt;
    X(1:2,i+1) = X(1:2,i) + dX;
    X(3,i+1) = at(i+1);

    %measurement
    z(i) = H(t(i))*X(:,i)+n(t(i));

    %residuals
    r(i) = z(i)-H(t(i))*Xhat(:,i);

    %covariance matrix
    P_i = P(:,:,i);
    Pdot = F*P_i + P_i*transpose(F) - ...
        1/(Vc^2*R1*(tf-t(i))^2+Vc^2*R2)*P_i...
     *transpose(Hbar)*Hbar*P_i+W;
    P(:,:,i+1) = Pdot*dt + P(:,:,i);

    %calculating Kalman gains
    K1(i+1) = P(1,1,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));
    K2(i+1) = P(1,2,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));
    K3(i+1) = P(1,3,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));

    %estimates
    K = [K1(i) K2(i) K3(i)]';
    dXhat = F*Xhat(:,i)*dt + K*(z(i)-H(t(i))*Xhat(:,i))*dt;
    Xhat(:,i+1) = Xhat(:,i) + dXhat;
end
Ehat = X - Xhat;
end
```

F Telegraph Kalman Filter (One Realization)

```
%Helene Levy
%MAE 271B Project
%Telegraphing
clc; clear; close all;

%% Given Parameters and Statistics
% time parameters
dt = 0.001;
tf = 10; % sec
tau = 2; % sec

t = 0:dt:tf;

% lateral position
m_y = 0;
var_y = 0;

% lateral velocity
m_v = 0;
var_v = 200^2; % (ft/sec)^2
corr_yv = 0;

% target acceleration
m_at = 0;
var_at = 100^2; % (ft/sec^2)^2

% target acceleration
m_aTbar = 0;
aT = 100; %ft/sec^2
RaT = @(t,s) aT^2*exp(-2*lambda*abs(t-s));

%relative velocity
```

```
R1 = 15*10^(-6); %rad^2/sec
R2 = 1.67*10^(-3); %rad^2/sec^3
Vc = 300; %ft/sec

% process noise spectral density
W = [ 0 0 0; 0 0 0; 0 0 var_at];

%fading and scintillation noise
m_n = 0;
V = @(t) R1 + R2/(tf-t)^2 ;

% initial covariance
P0 = [var_y 0 0; 0 var_v 0; 0 0 var_at];

%% Telegraph Process Generation
%poisson variable param
lambda = 0.25; %/sec

a =[];

%initial aT
A = rand;
if A ≤ 0.5
    a(1) = aT;
else
    a(1) = -aT;
end

%generating random switching times for telegraph signal
time = [];
time(1) = 0; i = 1;
t_np1 = 0;
%only want times in our simulation <10
```

```
while t_np1 < 10
    %uniform random variable
    U = rand;
    %next random switching time
    t_np1 = time(i) - 1/lambda*log(U);
    if t_np1 < 10
        time(i+1) = t_np1;
        a(i+1) = -a(i);
    end
    i = i+1;
end

%rounding the random time to accuracy of dt
time = round(time,-log10(dt));
t = round(t,-log10(dt));
[~,loc] = ismember(time,t);

%converting a to timescale
at = zeros(1,length(t));
for j = 2:length(loc)
    at(loc(j-1):loc(j)) = a(j-1);
end
at(loc(end):end) = a(end);

%visualizing telegraph signal
stairs(t,at)
ylim([-110,110])

%% Kalman Filter
% State space matrices
%two state telegraph
Ft = [0 1; 0 0];
Gt = [0; -1];
```

```
%original three state
F = [0 1 0; 0 0 -1; 0 0 -1/tau];
B = [0; 1; 0];
G = [0; 0; 1];
H = @(t) [1/(Vc*(tf-t)) 0 0];
Hbar = [1 0 0];

%initial states
y_0 = m_y + sqrt(var_y)*randn(1,1); %(0)
v_0 = m_v + sqrt(var_v)*randn(1,1);

% noise generation
n = @(t) m_n + sqrt(V(t)/dt)*randn(1);

X0 = [y_0 v_0 at(1)]';

%preallocation matrices
Xhat = zeros(3,length(t));
X = zeros(3,length(t));
P = zeros(3,3,length(t));
z = zeros(1,length(t)-1);
r = zeros(1,length(t)-1);

K1 = zeros(1,length(t));
K2 = zeros(1,length(t));
K3 = zeros(1,length(t));

X(:,1) = X0;
P(:,:,1) = P0;
K1(1) = P(1,1,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));
K2(1) = P(1,2,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));
K3(1) = P(1,3,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));
```

```matlab
a_p = 0;

for i = 1: length(t)-1
    %true states
    dX = (Ft*X(1:2,i)+Gt*at(i))*dt;
    X(1:2,i+1) = X(1:2,i) + dX;
    X(3,i+1) = at(i+1);

    %measurement
    z(i) = H(t(i))*X(:,i)+n(t(i));

    %covariance matrix
    P_i = P(:,:,i);
    Pdot = F*P_i + P_i*transpose(F) - ...
        1/(Vc^2*R1*(tf-t(i))^2+Vc^2*R2)*P_i...
     *Hbar'*Hbar*P_i+W;
    P(:,:,i+1) = Pdot*dt + P(:,:,i);

    %calculating Kalman gains
    K1(i+1) = P(1,1,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));
    K2(i+1) = P(1,2,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));
    K3(i+1) = P(1,3,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));

    %estimates
    K = [K1(i) K2(i) K3(i)]';
    dXhat = F*Xhat(:,i)*dt + K*(z(i)-H(t(i))*Xhat(:,i))*dt;
    Xhat(:,i+1) = Xhat(:,i) + dXhat;
end
Ehat = X - Xhat;
sig= sqrt([squeeze(P(1,1,:)), squeeze(P(2,2,:)), squeeze(P(3,3,:))]);

%% Plotting Results
%plotting Kalman gains vs. time
```

```
figure;
plot(t,K1,'b-'); hold on;
plot(t,K2,'r--'); hold on;
plot(t,K3,'m-.');

legend('K1','K2','K3');
xlabel('time (sec)');
ylabel('Kalman Filter Gain');
title('Filter Gain History');

figure;
rms_y = sqrt(squeeze(P(1,1,:)));
rms_v = sqrt(squeeze(P(2,2,:)));
rms_at = sqrt(squeeze(P(3,3,:)));
plot(t,rms_y,'b-'); hold on;
plot(t,rms_v,'r--'); hold on;
plot(t,rms_at,'m-.');

legend({'position (ft)','velocity (ft/sec)','acceleration (ft/sec^2)'});
xlabel('time (sec)');
ylabel('Standard deviation of the state error');
title('Evolution of Estimation Error RMS');

%plotting states
figure;
subplot(311)
plot(t,X(1,:)); hold on;
plot(t,Xhat(1,:))
legend('true', 'estimate');
title('Position');
xlabel('time (s)');

subplot(312)
```

```
plot(t,X(2,:)); hold on;
plot(t,Xhat(2,:))
legend('true', 'estimate');
title('Velocity');
xlabel('time (s)');

subplot(313)
plot(t,X(3,:)); hold on;
plot(t,Xhat(3,:))
legend('true', 'estimate');
title('Target Acceleration');
xlabel('time (s)');

%plotting state errors

figure;
subplot(311)
stairs(t,Ehat(1,:));hold on;
plot(t,sig(:,1),'r--',t,-sig(:,1),'r--');
title('Position Error');
xlabel('time (s)');
ylabel('position error');

subplot(312)
stairs(t,Ehat(2,:)); hold on;
plot(t,sig(:,2),'r--',t,-sig(:,2),'r--');
title('Velocity Error');
xlabel('time (s)');
ylabel('velocity error');

subplot(313)
stairs(t,Ehat(3,:)); hold on;
plot(t,sig(:,3),'r--',t,-sig(:,3),'r--');
```

```
title('Target Acceleration Error');
xlabel('time (s)');
ylabel('acceleration error');
```