

## 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.

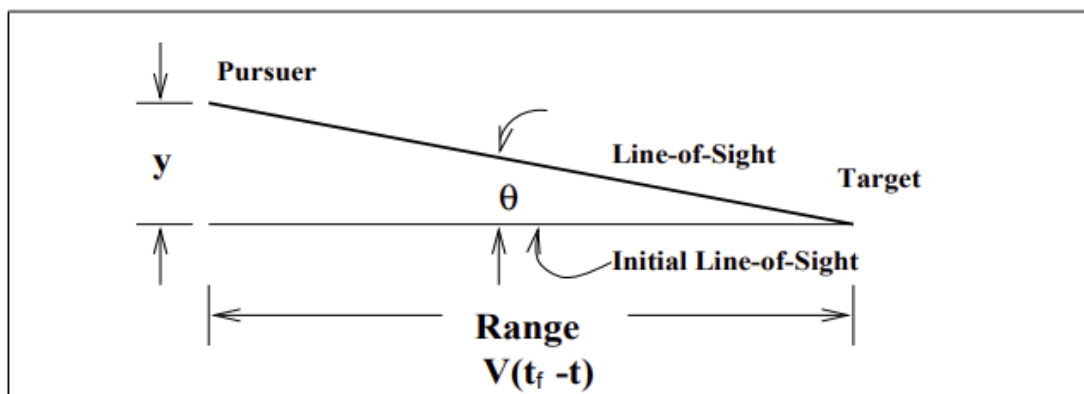


Figure 1: Missile Intercept Illustration

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,  $\tau$ , 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,  $\theta$ . 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

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

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

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

The initial covariance is

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

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:

$$\begin{aligned} \dot{x} &= Fx(t) + Ba_P + Gw_{a_T} \\ \begin{Bmatrix} \dot{y} \\ \dot{v} \\ \dot{a_T} \end{Bmatrix} &= \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & -\frac{1}{\tau} \end{bmatrix}}_F \underbrace{\begin{Bmatrix} y \\ v \\ a_T \end{Bmatrix}}_x + \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}}_B a_P + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_G w_{a_T} \end{aligned} \quad (5)$$

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  $\theta$ . Using Equations (2) and (3), for  $\theta$  and  $z$ , the state space model of the measurement can be summarized in the following Equation (6).

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

with initial states as follows:

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

## 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  $\lambda$  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

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

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,

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

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.

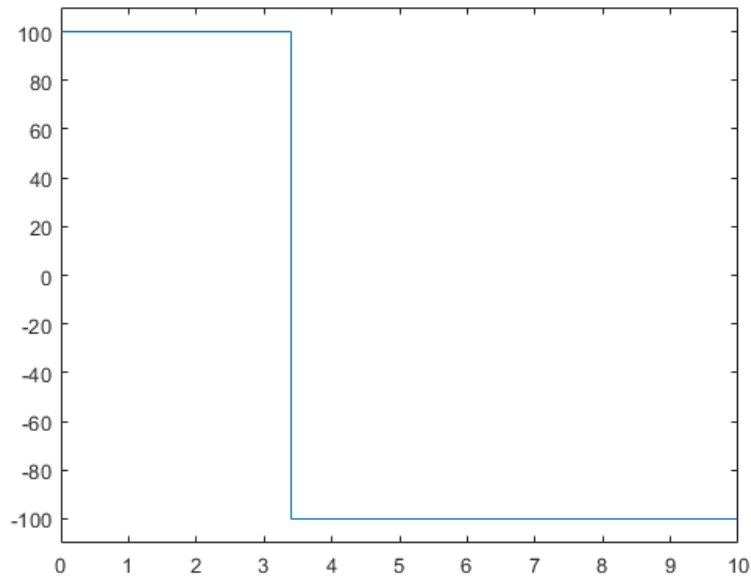


Figure 2: Random Telegraph Target Acceleration

### 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).

$$\dot{x} = F_t x(t) + G_t a_T$$

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

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).

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

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

### 3.3 Continuous-Time Kalman Filter

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

$$dx(t) = F(t)x(t)dt + G(t)d\beta_t \quad (10)$$

$$dz(t) = H(t)x(t)dt + dn \quad (11)$$

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:

$$d\hat{x}(t) = F(t)\hat{x}(t)dt + K(t)[dz(t) - H(t)\hat{x}(t)dt] \quad (12)$$

$$K(t) = P(t)H(t)^TV(t)^{-1}$$

covariance update (Riccati Equation):

$$\dot{P}(t) = F(t)P(t) + P(t)F(t)^T - P(t)H(t)^TV(t)^{-1}H(t)P(t) + G(t)W(t)G(t)^T$$

$$P(0) = P_0$$

For the missile intercept problem the Riccati equation becomes

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

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:

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

### 3.4 Validation Analysis

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

$$E[r_k r_j^T] = 0 \quad \forall j < k \quad (15)$$

#### 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  $t_i \in [0, 30]$ .

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

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.

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

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

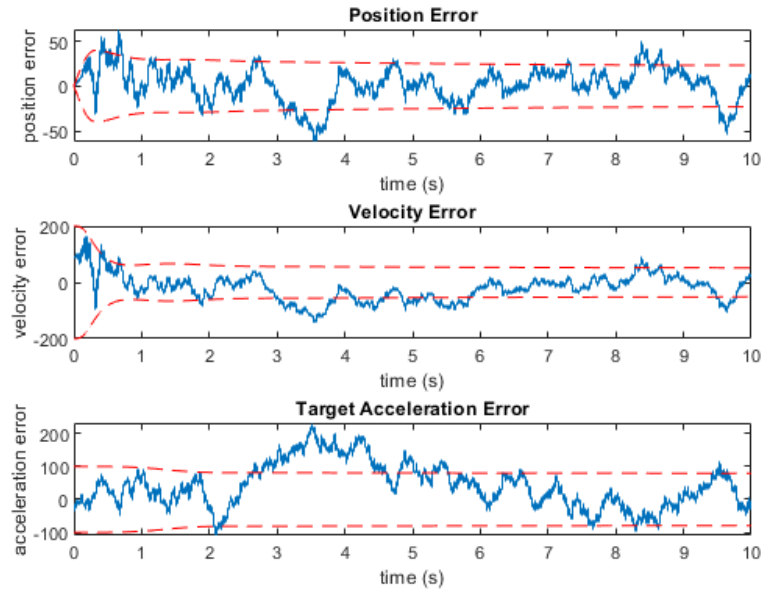
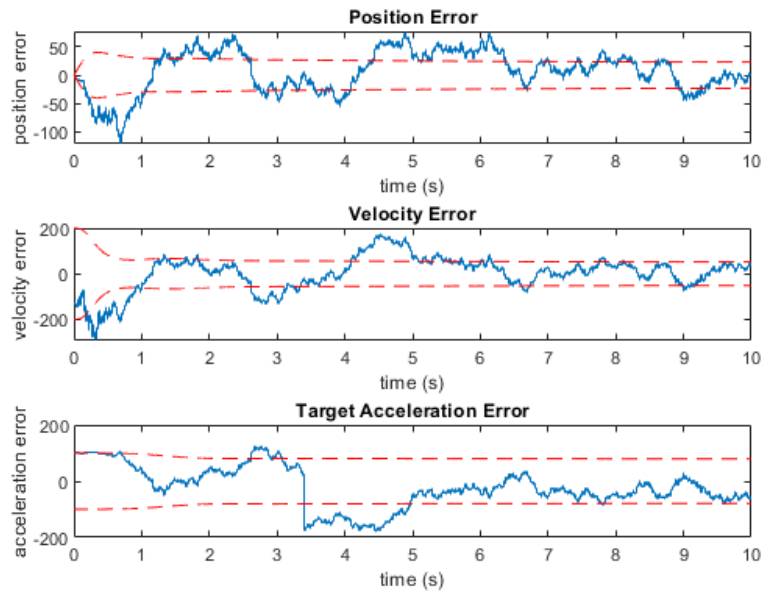
independence of residuals check:

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

## 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.

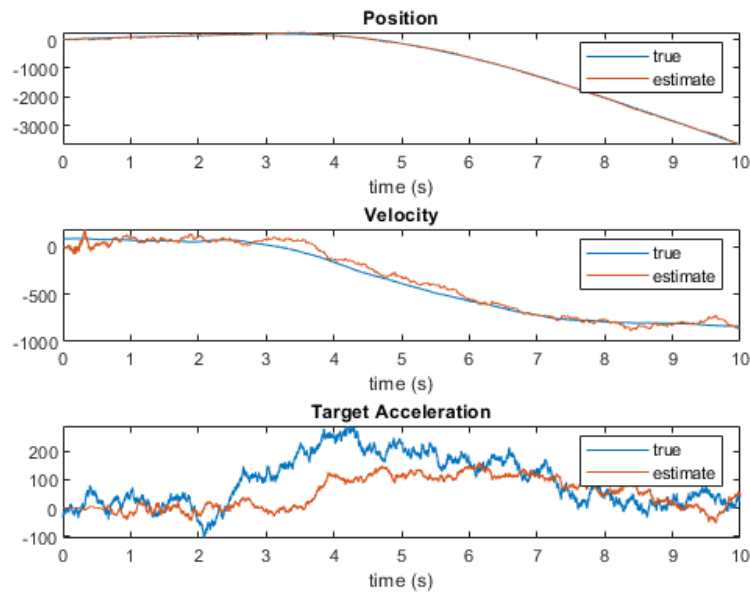


Figure 5: G-M State Estimate,  $\hat{x}(t_i)$

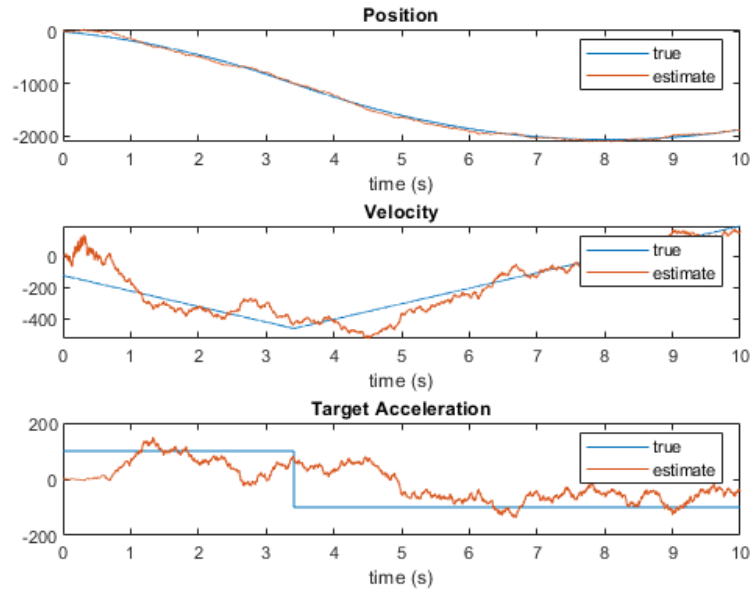


Figure 6: Telegraph State Estimate,  $\hat{x}(t_i)$

## 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.

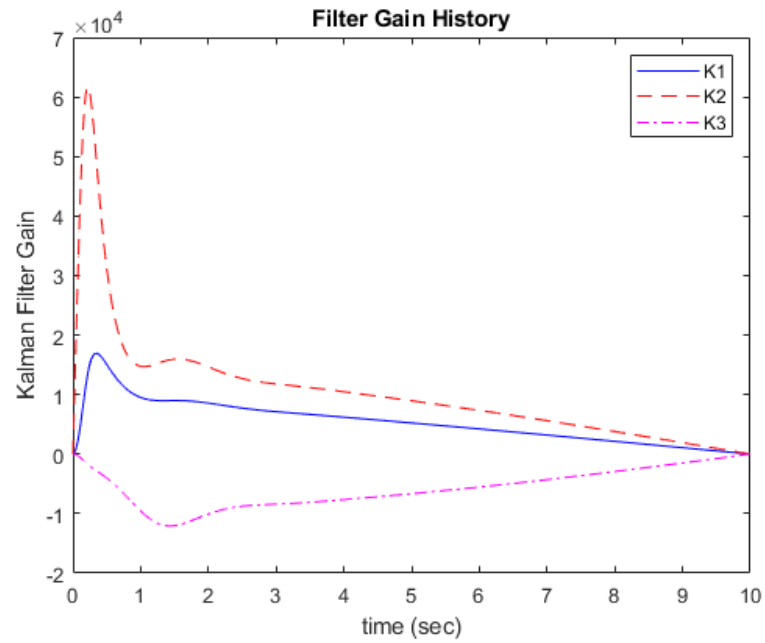


Figure 7: Evolution of Kalman Gain

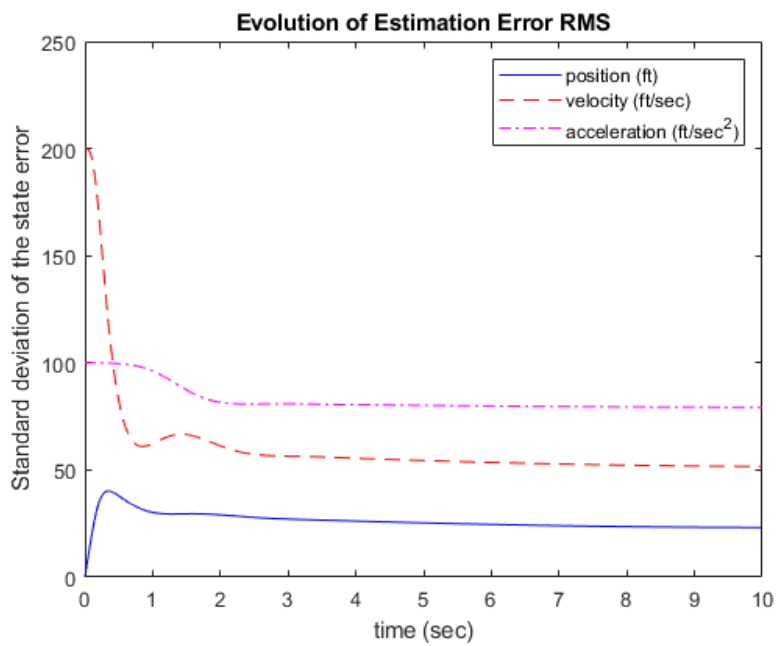
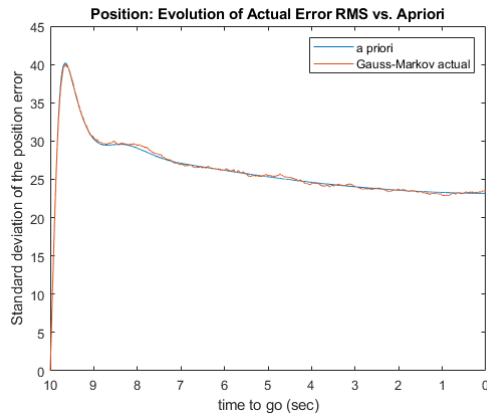


Figure 8: Evolution of Estimation Error RMS

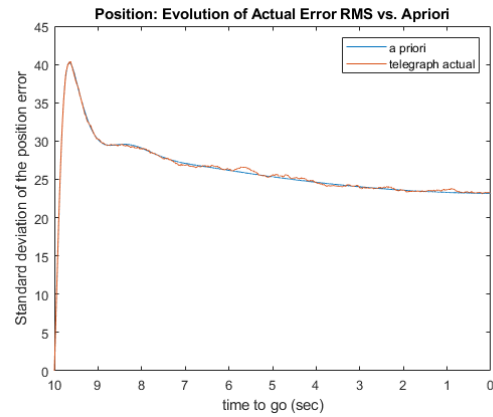
### 4.3 Filter Validation and Simulation

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.

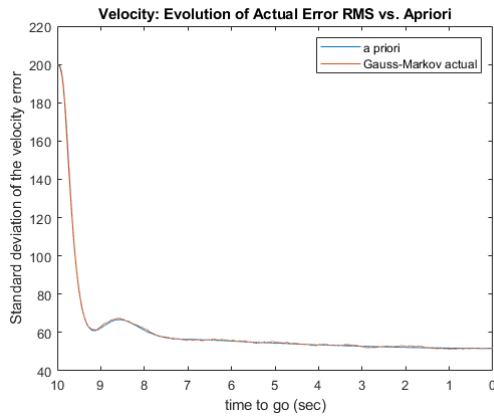
First, the filter was validated by comparing the simulated error variance,  $P^{ave}$  found with Equation (17), with the filter error variance derived from one realization,  $P$ . Figure 9 shows the comparison between the standard deviation of the state errors of both the models ( $\sqrt{P^{ave}}$  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.



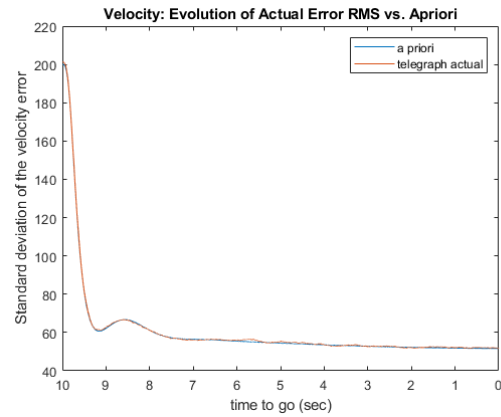
(a) G-M Lateral Position STD



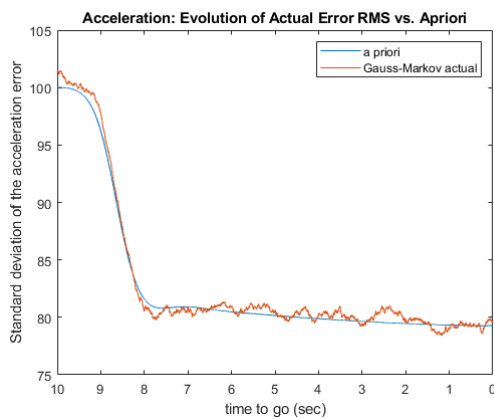
(b) Telegraph Lateral Position STD



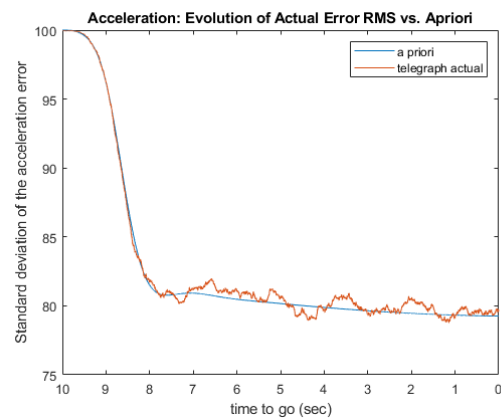
(c) G-M Lateral Velocity STD



(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

```
1 %Helene Levy
2 %MAE 271B Project
3 %Monte Carlo Simulation
4 clc; clear; close all;
5
6 %time step choices
7 dt = 0.001;
8 tf = 10;
9 t = 0:dt:tf;
10
11 %preallocations
12 e_sum = zeros(3,length(t));
13 e_ave = zeros(3,length(t));
14 r_sum = zeros(1,1);
15
16 P_ave = zeros(3,3,length(t));
17 P_sum = zeros(3,3,length(t));
18 % ortho_ave = zeros(3,3,length(t));
19 % ortho_sum = zeros(3,3,length(t));
20
21 %iterating through N times
22 N = 10000;
23 for j = 1:N
24     [X_hat,e_hat,P,r] = ct_kalman_filter(dt);
25
26     %ensemble average of error
27     e_sum = e_sum + e_hat;
28     e_ave = 1/j*e_sum;
29
30     for k = 1:length(e_hat)
```

```

31     %actual error variance calculation
32     P_sum(:, :, k) = P_sum(:, :, k) + (e_hat(:, k) - e_ave(:, k)) ...
33         * (e_hat(:, k) - e_ave(:, k))';
34     P_ave(:, :, k) = 1 / (j - 1) * P_sum(:, :, k);
35
36     %     ortho_sum(:, :, k) = ...
37     ortho_sum(:, :, k) + (e_hat(:, k) - e_ave(:, k)) * X_hat(:, k)';
38     %     ortho_ave(:, :, k) = 1 / j * ortho_sum(:, :, k);
39     end
40
41     %ensemble average for correlation of residuals at end time
42     r_sum = r_sum + r(end / 2) * (r(end / 2 - 1))';
43     r_ave = 1 / j * r_sum;
44     end
45
46     %showing residuals uncorrelated
47     disp('ensemble average for correlation of the residuals:');
48     disp(r_ave);
49
50     %apriori standard deviation and actual standard deviation
51     std_true = sqrt([squeeze(P(1, 1, :)), squeeze(P(2, 2, :)), squeeze(P(3, 3, :))]);
52     std_ave = sqrt([squeeze(P_ave(1, 1, :)), squeeze(P_ave(2, 2, :)), ...
53         squeeze(P_ave(3, 3, :))]);
54
55     %plotting std comparison
56     figure;
57     plot(tf - t, std_true(:, 1)); hold on;
58     plot(tf - t, std_ave(:, 1));
59     set(gca, 'xdir', 'reverse');
60     xlabel('time to go (sec)');
61     ylabel('Standard deviation of the position error');
62     title('Position: Evolution of Actual Error RMS vs. Apriori');
63     legend('a priori', 'Gauss-Markov actual')

```

```
63
64 figure;
65 plot(tf-t,std_true(:,2)); hold on;
66 plot(tf-t,std_ave(:,2));
67 set ( gca, 'xdir', 'reverse' );
68 xlabel('time to go (sec)');
69 ylabel('Standard deviation of the velocity error');
70 title('Velocity: Evolution of Actual Error RMS vs. Apriori')
71 legend('a priori','Gauss-Markov actual')
72
73 figure;
74 plot(tf-t,std_true(:,3)); hold on;
75 plot(tf-t,std_ave(:,3));
76 set ( gca, 'xdir', 'reverse' );
77 xlabel('time to go (sec)');
78 ylabel('Standard deviation of the acceleration error');
79 title('Acceleration: Evolution of Actual Error RMS vs. Apriori')
80 legend('a priori','Gauss-Markov actual')
81
82 % cross terms in covariance matrix plotting
83 % figure;
84 % for i = 1:size(P_ave,1)
85 %     for k = 1:size(P_ave,2)
86 %         dP = squeeze(P_ave(i,k,:))-squeeze(P(i,k,:));
87 %         plot(t,dP); hold on;
88 %     end
89 % end
90 %
91 % legend('P(1,1)', 'P(1,2)', 'P(1,3)', 'P(2,1)', 'P(2,2)', 'P(2,3)', 'P(3,1)', ...
92 %         'P(3,2)', 'P(3,3)', 'Location', 'southeast');
93 % title('Simulated - Filter Error Variance, P_{ave} - P');
94 % xlabel('time (s)');
95 % ylabel('P_{ave} - P');
```

## B G-M Kalman Filter Function

```
1 function [Xhat,Ehat,P,r] = ct_kalman_filter(dt)
2 % function for running continuous time kalman filter
3
4 % Given parameters and statistics
5
6 % time parameters
7 tf = 10; % sec
8 tau = 2; % sec
9
10 R1 = 15*10^(-6); %rad^2/sec
11 R2 = 1.67*10^(-3); %rad^2/sec^3
12 Vc = 300; %ft/sec
13
14 % target acceleration
15 m_at = 0;
16 var_at = 100^2; % (ft/sec^2)^2
17
18 % lateral position
19 m_y = 0;
20 var_y = 0;
21
22 % lateral velocity
23 m_v = 0;
24 var_v = 200^2; % (ft/sec)^2
25
26 %fading and scintillation noise
27 m_n = 0;
28 V = @(t) R1 + R2/(tf-t)^2 ;
29
30 % process noise spectral density
```

```
31 W = [ 0 0 0; 0 0 0; 0 0 var_at];
32
33 % initial covariance
34 P0 = [var_y 0 0; 0 var_v 0; 0 0 var_at];
35
36 % State Space Matrices
37 % xdot = Fc + Bap + Gw_at
38 % x = [y v at]
39 F = [0 1 0; 0 0 -1; 0 0 -1/tau];
40 B = [0; 1; 0];
41 G = [0; 0; 1];
42
43 H = @(t) [1/(Vc*(tf - t)) 0 0];
44 Hbar = [1 0 0];
45
46 %K and P calculations using ode45
47 tspan = 0:dt:tf;
48 [t,P] = ode45(@Pdot, tspan , P0(:));
49 P = reshape(P.',3,3,[]);
50
51 %calculating Kalman gains
52 K1 = squeeze(P(1,1,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));
53 K2 = squeeze(P(1,2,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));
54 K3 = squeeze(P(1,3,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));
55
56 %initial states
57 y_0 = m_y + sqrt(var_y)*randn(1,1); %(0)
58 v_0 = m_v + sqrt(var_v)*randn(1,1);
59 at_0 = m_at + sqrt(var_at)*randn(1,1);
60
61 % noise generation
62 w_at = m_at + sqrt(var_at/dt)*randn(1,length(t)-1);
63 n = @(t) m_n + sqrt(V(t)/dt)*randn(1);
```

```
64
65 %preallocating matrices
66 Xhat = zeros(3,length(t));
67 X = zeros(3,length(t));
68 z = zeros(1,length(t)-1);
69 r = zeros(1,length(t)-1);
70
71 X0 = [y_0 v_0 at_0]';
72
73 %initial conditions of state
74 X(:,1) = X0;
75 a_p = 0;
76 for i = 1: length(t)-1
77     %true states
78     dX = (F*X(:,i)+B*a_p+G*w_at(i))*dt;
79     X(:,i+1) = X(:,i) + dX;
80
81     %measurement
82     z(i) = H(t(i))*X(:,i)+n(t(i));
83
84     %residuals
85     r(i) = z(i)-H(t(i))*Xhat(:,i);
86
87     %estimates
88     K = [K1(i) K2(i) K3(i)]';
89     dXhat = F*Xhat(:,i)*dt + K*(z(i)-H(t(i))*Xhat(:,i))*dt;
90     Xhat(:,i+1) = Xhat(:,i) + dXhat;
91
92 end
93 Ehat = X - Xhat;
94 end
```

## C G-M Kalman Filter (One Realization)

```
1 % MAE 271B Project
2 % Helene Levy
3 clc; clear; close all;
4
5 %% Given parameters and statistics
6
7 % time parameters
8 tf = 10; % sec
9 tau = 2; % sec
10
11 R1 = 15*10^(-6); %rad^2/sec
12 R2 = 1.67*10^(-3); %rad^2/sec^3
13 Vc = 300; %ft/sec
14
15 % target acceleration
16 m_at = 0;
17 var_at = 100^2; % (ft/sec^2)^2
18 corr_at_as = @(t,s) var_at*exp(-(t-s)/tau);
19
20 % lateral position
21 m_y = 0;
22 var_y = 0;
23
24 % lateral velocity
25 m_v = 0;
26 var_v = 200^2; % (ft/sec)^2
27 corr_yv = 0;
28
29 %fading and scintillation noise
```

```
30 m_n = 0;
31 V = @(t) R1 + R2/(tf-t)^2 ;
32 corr_nt_tau = @(t) (R1 + R2/(tf-t)^2)*dirac(t-tau);
33
34 % process noise spectral density
35 W = [ 0 0 0; 0 0 0; 0 0 var_at];
36
37 % initial covariance
38 P0 = [var_y 0 0; 0 var_v 0; 0 0 var_at];
39
40
41 %% State Space Matrices
42 % xdot = Fc + Bap + Gw_at
43 % x = [y v at]
44 F = [0 1 0; 0 0 -1; 0 0 -1/tau];
45 B = [0; 1; 0];
46 G = [0; 0; 1];
47
48 H = @(t) [1/(Vc*(tf - t)) 0 0];
49 Hbar = [1 0 0];
50
51 %% P, K calculation and Plotting Figures 9.6 and 9.7
52 dt = 0.0001;
53 tspan = 0:dt:tf;
54 [t,P] = ode45(@Pdot, tspan , P0(:));
55 P = reshape(P.',3,3,[]);
56
57 %calculating Kalman gains
58 K1 = squeeze(P(1,1,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));
59 K2 = squeeze(P(1,2,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));
60 K3 = squeeze(P(1,3,:))./(Vc*R1.*(tf-t) + Vc*R2./(tf-t));
61
62 %plotting Kalman gains vs. time
```



```
63 figure;
64 plot(t,K1,'b-'); hold on;
65 plot(t,K2,'r--'); hold on;
66 plot(t,K3,'m-.');
67
68 legend('K1','K2','K3');
69 xlabel('time (sec)');
70 ylabel('Kalman Filter Gain');
71 title('Filter Gain History');
72
73 figure;
74 rms_y = sqrt(squeeze(P(1,1,:)));
75 rms_v = sqrt(squeeze(P(2,2,:)));
76 rms_at = sqrt(squeeze(P(3,3,:)));
77 plot(t,rms_y,'b-'); hold on;
78 plot(t,rms_v,'r--'); hold on;
79 plot(t,rms_at,'m-.');
80
81 legend({'position (ft)','velocity (ft/sec)','acceleration (ft/sec^2)'});
82 xlabel('time (sec)');
83 ylabel('Standard deviation of the state error');
84 title('Evolution of Estimation Error RMS');
85
86 %% Kalman Filter
87 %initial states
88 y_0 = m_y + sqrt(var_y)*randn(1,1); %(0)
89 v_0 = m_v + sqrt(var_v)*randn(1,1);
90 at_0 = m_at + sqrt(var_at)*randn(1,1);
91
92 % noise generation
93 w_at = m_at + sqrt(var_at/dt)*randn(1,length(t)-1);
94 n = @(t) m_n + sqrt(V(t)/dt)*randn(1);
95
```

```
96 X0 = [y_0 v_0 at_0]';
97
98 Xhat = zeros(3,length(t));
99
100 X = zeros(3,length(t));
101 X(:,1) = X0;
102 a_p = 0;
103
104 for i = 1: length(t)-1
105     %true states
106     dX = (F*X(:,i)+B*a_p+G*w_at(i))*dt;
107     X(:,i+1) = X(:,i) + dX;
108
109     %measurement
110     z = H(t(i))*X(:,i)+n(t(i));
111
112     %estimates
113     K = [K1(i) K2(i) K3(i)]';
114     dXhat = F*Xhat(:,i)*dt + K*(z-H(t(i))*Xhat(:,i))*dt;
115     Xhat(:,i+1) = Xhat(:,i) + dXhat;
116 end
117 Ehat = X - Xhat;
118 sig= sqrt([squeeze(P(1,1,:)), squeeze(P(2,2,:)), squeeze(P(3,3,:))]);
119
120 %plotting states
121 figure;
122 subplot(311)
123 plot(t,X(1,:)); hold on;
124 plot(t,Xhat(1,:))
125 legend('true', 'estimate');
126 title('Position');
127 xlabel('time (s)');
128
```

```
129 subplot(312)
130 plot(t,X(2,:)); hold on;
131 plot(t,Xhat(2,:))
132 legend('true', 'estimate');
133 title('Velocity');
134 xlabel('time (s)');
135
136 subplot(313)
137 plot(t,X(3,:)); hold on;
138 plot(t,Xhat(3,:))
139 legend('true', 'estimate');
140 title('Target Acceleration');
141 xlabel('time (s)');
142
143 %plotting state errors
144
145 figure;
146 subplot(311)
147 stairs(t,Ehat(1,:));hold on;
148 plot(t,sig(:,1),'r--',t,-sig(:,1),'r--');
149 title('Position Error');
150 xlabel('time (s)');
151 ylabel('position error');
152
153 subplot(312)
154 stairs(t,Ehat(2,:)); hold on;
155 plot(t,sig(:,2),'r--',t,-sig(:,2),'r--');
156 title('Velocity Error');
157 xlabel('time (s)');
158 ylabel('velocity error');
159
160 subplot(313)
161 stairs(t,Ehat(3,:)); hold on;
```

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

## D Telegraph Monte Carlo Simulation

```
1 %Helene Levy
2 %MAE 271B Project
3 %Monte Carlo Simulation
4 clc; clear; close all;
5
6 %time step choices
7 dt = 0.001;
8 tf = 10;
9 t = 0:dt:tf;
10
11 %preallocations
12 e_sum = zeros(3,length(t));
13 e_ave = zeros(3,length(t));
14 r_sum = zeros(1,1);
15
16 P_ave = zeros(3,3,length(t));
17 P_sum = zeros(3,3,length(t));
18
19 %iterating through N times
20 N = 10000;
21 for j = 1:N
22     [X_hat,e_hat,P,r] = tele_kalman_filt(dt);
23
```

```
24     %ensemble average of error
25     e_sum = e_sum + e_hat;
26     e_ave = 1/j*e_sum;
27
28     for k = 1:length(e_hat)
29         %actual error variance calculation
30         P_sum(:, :, k) = P_sum(:, :, k) + (e_hat(:, k) - e_ave(:, k)) ...
31             *(e_hat(:, k) - e_ave(:, k))';
32         P_ave(:, :, k) = 1/(j-1)*P_sum(:, :, k);
33
34     end
35
36     %ensemble average for correlation of residuals at end time
37     r_sum = r_sum + r(end/2)*(r(end/2-1))';
38     r_ave = 1/j*r_sum;
39 end
40
41 %showing residuals uncorrelated
42 disp('ensemble average for correlation of the residuals:');
43 disp(r_ave);
44
45 %apriori standard deviation and actual standard deviation
46 std_true = sqrt([squeeze(P(1,1,:)), squeeze(P(2,2,:)), squeeze(P(3,3,:))]);
47 std_ave = sqrt([squeeze(P_ave(1,1,:)), squeeze(P_ave(2,2,:)), ...
48     squeeze(P_ave(3,3,:))]);
49
50 %plotting std comparison
51 figure;
52 plot(tf-t, std_true(:,1)); hold on;
53 plot(tf-t, std_ave(:,1));
54 set ( gca, 'xdir', 'reverse' );
55 xlabel('time to go (sec)');
56 ylabel('Standard deviation of the position error');
```

```
57 title('Position: Evolution of Actual Error RMS vs. Apriori')
58 legend('a priori','telegraph actual')
59
60 figure;
61 plot(tf-t,std_true(:,2)); hold on;
62 plot(tf-t,std_ave(:,2));
63 set ( gca, 'xdir', 'reverse' );
64 xlabel('time to go (sec)');
65 ylabel('Standard deviation of the velocity error');
66 title('Velocity: Evolution of Actual Error RMS vs. Apriori')
67 legend('a priori','telegraph actual')
68
69 figure;
70 plot(tf-t,std_true(:,3)); hold on;
71 plot(tf-t,std_ave(:,3));
72 set ( gca, 'xdir', 'reverse' );
73 xlabel('time to go (sec)');
74 ylabel('Standard deviation of the acceleration error');
75 title('Acceleration: Evolution of Actual Error RMS vs. Apriori')
76 legend('a priori','telegraph actual')
```

## E Telegraph Kalman Filter Function

```
1 function [Xhat,Ehat,P,r] = tele_kalman_filt(dt)
2 % Given Parameters and Statistics
3 % time parameters
4 tf = 10; % sec
5 tau = 2; % sec
6 t = 0:dt:tf;
7
8 % lateral position
9 m_y = 0;
10 var_y = 0;
11
12 % lateral velocity
13 m_v = 0;
14 var_v = 200^2; % (ft/sec)^2
15 corr_yv = 0;
16
17 % target acceleration
18 m_at = 0;
19 var_at = 100^2; % (ft/sec^2)^2
20
21 % target acceleration
22 m_aTbar = 0;
23 aT = 100; %ft/sec^2
24 RaT = @(t,s) aT^2*exp(-2*lambda*abs(t-s));
25
26 %relative velocity
27 R1 = 15*10^(-6); %rad^2/sec
28 R2 = 1.67*10^(-3); %rad^2/sec^3
29 Vc = 300; %ft/sec
30
```

```
31 % process noise spectral density
32 W = [ 0 0 0; 0 0 0; 0 0 var_at];
33
34 %fading and scintillation noise
35 m_n = 0;
36 V = @(t) R1 + R2/(tf-t)^2 ;
37
38 % initial covariance
39 P0 = [var_y 0 0; 0 var_v 0; 0 0 var_at];
40
41 %Telegraph Process Generation
42 %poisson variable param
43 lambda = 0.25; %/sec
44
45 a = [];
46
47 %initial aT
48 A = rand;
49 if A ≤ 0.5
50     a(1) = aT;
51 else
52     a(1) = -aT;
53 end
54
55 %generating random switching times for telegraph signal
56 time = [];
57 time(1) = 0; i = 1;
58 t_np1 = 0;
59 %only want times in our simulation <10
60 while t_np1 < 10
61     %uniform random variable
62     U = rand;
63     %next random switching time
```



```
64     t_npl = time(i) - 1/lambda*log(U);
65     if t_npl < 10
66         time(i+1) = t_npl;
67         a(i+1) = -a(i);
68     end
69     i = i+1;
70 end
71
72 %rounding the random time to accuracy of dt
73 time = round(time,-log10(dt));
74 t = round(t,-log10(dt));
75 [~,loc] = ismember(time,t);
76
77 %converting a to timescale
78 at = zeros(1,length(t));
79 for j = 2:length(loc)
80     at(loc(j-1):loc(j)) = a(j-1);
81 end
82 at(loc(end):end) = a(end);
83
84 % %visualizing telegraph signal
85 % stairs(t,at)
86
87 % Filter
88 % State space matrices
89 %two state telegraph
90 Ft = [0 1; 0 0];
91 Gt = [0; -1];
92
93 %original three state
94 F = [0 1 0; 0 0 -1; 0 0 -1/tau];
95 B = [0; 1; 0];
96 G = [0; 0; 1];
```

```
97 H = @(t) [1/(Vc*(tf-t)) 0 0];
98 Hbar = [1 0 0];
99
100 %initial states
101 y_0 = m_y + sqrt(var_y)*randn(1,1); %(0)
102 v_0 = m_v + sqrt(var_v)*randn(1,1);
103
104 % noise generation
105 n = @(t) m_n + sqrt(V(t)/dt)*randn(1);
106
107 X0 = [y_0 v_0 at(1)]';
108
109 %preallocation matrices
110 Xhat = zeros(3,length(t));
111 X = zeros(3,length(t));
112 z = zeros(1,length(t)-1);
113 r = zeros(1,length(t)-1);
114
115 P = zeros(3,3,length(t));
116
117 K1 = zeros(1,length(t));
118 K2 = zeros(1,length(t));
119 K3 = zeros(1,length(t));
120
121 %intitital values
122 X(:,1) = X0;
123 P(:, :, 1) = P0;
124 K1(1) = P(1,1,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));
125 K2(1) = P(1,2,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));
126 K3(1) = P(1,3,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));
127
128 for i = 1: length(t)-1
129     %true states
```

```

130     dX = (Ft*X(1:2,i)+Gt*at(i))*dt;
131     X(1:2,i+1) = X(1:2,i) + dX;
132     X(3,i+1) = at(i+1);
133
134     %measurement
135     z(i) = H(t(i))*X(:,i)+n(t(i));
136
137     %residuals
138     r(i) = z(i)-H(t(i))*Xhat(:,i);
139
140     %covariance matrix
141     P_i = P(:, :, i);
142     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;
144     P(:, :, i+1) = Pdot*dt + P(:, :, i);
145
146     %calculating Kalman gains
147     K1(i+1) = P(1,1,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));
148     K2(i+1) = P(1,2,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));
149     K3(i+1) = P(1,3,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));
150
151     %estimates
152     K = [K1(i) K2(i) K3(i)]';
153     dXhat = F*Xhat(:,i)*dt + K*(z(i)-H(t(i))*Xhat(:,i))*dt;
154     Xhat(:,i+1) = Xhat(:,i) + dXhat;
155 end
156 Ehat = X - Xhat;
157 end

```

## F Telegraph Kalman Filter (One Realization)

```
1 %Helene Levy
2 %MAE 271B Project
3 %Telegraphing
4 clc; clear; close all;
5
6 %% Given Parameters and Statistics
7 % time parameters
8 dt = 0.001;
9 tf = 10; % sec
10 tau = 2; % sec
11
12 t = 0:dt:tf;
13
14 % lateral position
15 m_y = 0;
16 var_y = 0;
17
18 % lateral velocity
19 m_v = 0;
20 var_v = 200^2; % (ft/sec)^2
21 corr_yv = 0;
22
23 % target acceleration
24 m_at = 0;
25 var_at = 100^2; % (ft/sec^2)^2
26
27 % target acceleration
28 m_aTbar = 0;
29 aT = 100; %ft/sec^2
30 RaT = @(t,s) aT^2*exp(-2*lambda*abs(t-s));
31
32 %relative velocity
```

```
33 R1 = 15*10^(-6); %rad^2/sec
34 R2 = 1.67*10^(-3); %rad^2/sec^3
35 Vc = 300; %ft/sec
36
37 % process noise spectral density
38 W = [ 0 0 0; 0 0 0; 0 0 var_at];
39
40 %fading and scintillation noise
41 m_n = 0;
42 V = @(t) R1 + R2/(tf-t)^2 ;
43
44 % initial covariance
45 P0 = [var_y 0 0; 0 var_v 0; 0 0 var_at];
46
47 %% Telegraph Process Generation
48 %poisson variable param
49 lambda = 0.25; %/sec
50
51 a = [];
52
53 %initial aT
54 A = rand;
55 if A ≤ 0.5
56     a(1) = aT;
57 else
58     a(1) = -aT;
59 end
60
61 %generating random switching times for telegraph signal
62 time = [];
63 time(1) = 0; i = 1;
64 t_npl = 0;
65 %only want times in our simulation <10
```

```
66 while t_npl < 10
67     %uniform random variable
68     U = rand;
69     %next random switching time
70     t_npl = time(i) - 1/lambda*log(U);
71     if t_npl < 10
72         time(i+1) = t_npl;
73         a(i+1) = -a(i);
74     end
75     i = i+1;
76 end
77
78 %rounding the random time to accuracy of dt
79 time = round(time,-log10(dt));
80 t = round(t,-log10(dt));
81 [~,loc] = ismember(time,t);
82
83 %converting a to timescale
84 at = zeros(1,length(t));
85 for j = 2:length(loc)
86     at(loc(j-1):loc(j)) = a(j-1);
87 end
88 at(loc(end):end) = a(end);
89
90 %visualizing telegraph signal
91 stairs(t,at)
92 ylim([-110,110])
93
94 %% Kalman Filter
95 % State space matrices
96 %two state telegraph
97 Ft = [0 1; 0 0];
98 Gt = [0; -1];
```

```

99
100 %original three state
101 F = [0 1 0; 0 0 -1; 0 0 -1/tau];
102 B = [0; 1; 0];
103 G = [0; 0; 1];
104 H = @(t) [1/(Vc*(tf-t)) 0 0];
105 Hbar = [1 0 0];
106
107 %initial states
108 y_0 = m_y + sqrt(var_y)*randn(1,1); %(0)
109 v_0 = m_v + sqrt(var_v)*randn(1,1);
110
111 % noise generation
112 n = @(t) m_n + sqrt(V(t)/dt)*randn(1);
113
114 X0 = [y_0 v_0 at(1)]';
115
116 %preallocation matrices
117 Xhat = zeros(3,length(t));
118 X = zeros(3,length(t));
119 P = zeros(3,3,length(t));
120 z = zeros(1,length(t)-1);
121 r = zeros(1,length(t)-1);
122
123 K1 = zeros(1,length(t));
124 K2 = zeros(1,length(t));
125 K3 = zeros(1,length(t));
126
127 X(:,1) = X0;
128 P(:, :, 1) = P0;
129 K1(1) = P(1,1,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));
130 K2(1) = P(1,2,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));
131 K3(1) = P(1,3,1)/(Vc*R1*(tf-t(1)) + Vc*R2/(tf-t(1)));

```

```
132 a_p = 0;
133
134 for i = 1: length(t)-1
135     %true states
136     dX = (Ft*X(1:2,i)+Gt*at(i))*dt;
137     X(1:2,i+1) = X(1:2,i) + dX;
138     X(3,i+1) = at(i+1);
139
140     %measurement
141     z(i) = H(t(i))*X(:,i)+n(t(i));
142
143     %covariance matrix
144     P_i = P(:, :, i);
145     Pdot = F*P_i + P_i*transpose(F) - ...
146           1/(Vc^2*R1*(tf-t(i))^2+Vc^2*R2)*P_i...
147           *Hbar'*Hbar*P_i+W;
148     P(:, :, i+1) = Pdot*dt + P(:, :, i);
149
150     %calculating Kalman gains
151     K1(i+1) = P(1,1,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));
152     K2(i+1) = P(1,2,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));
153     K3(i+1) = P(1,3,i+1)/(Vc*R1*(tf-t(i+1)) + Vc*R2/(tf-t(i+1)));
154
155     %estimates
156     K = [K1(i) K2(i) K3(i)]';
157     dXhat = F*Xhat(:,i)*dt + K*(z(i)-H(t(i))*Xhat(:,i))*dt;
158     Xhat(:,i+1) = Xhat(:,i) + dXhat;
159 end
160 Ehat = X - Xhat;
161
162 sig= sqrt([squeeze(P(1,1,:)), squeeze(P(2,2,:)), squeeze(P(3,3,:))]);
163
164 %% Plotting Results
165 %plotting Kalman gains vs. time
```



```
164 figure;
165 plot(t,K1,'b-'); hold on;
166 plot(t,K2,'r--'); hold on;
167 plot(t,K3,'m-.');
168
169 legend('K1','K2','K3');
170 xlabel('time (sec)');
171 ylabel('Kalman Filter Gain');
172 title('Filter Gain History');
173
174 figure;
175 rms_y = sqrt(squeeze(P(1,1,:)));
176 rms_v = sqrt(squeeze(P(2,2,:)));
177 rms_at = sqrt(squeeze(P(3,3,:)));
178 plot(t,rms_y,'b-'); hold on;
179 plot(t,rms_v,'r--'); hold on;
180 plot(t,rms_at,'m-.');
181
182 legend({'position (ft)','velocity (ft/sec)','acceleration (ft/sec^2)'});
183 xlabel('time (sec)');
184 ylabel('Standard deviation of the state error');
185 title('Evolution of Estimation Error RMS');
186
187 %plotting states
188 figure;
189 subplot(311)
190 plot(t,X(1,:)); hold on;
191 plot(t,Xhat(1,:))
192 legend('true','estimate');
193 title('Position');
194 xlabel('time (s)');
195
196 subplot(312)
```

```
197 plot(t,X(2,:)); hold on;
198 plot(t,Xhat(2,:))
199 legend('true', 'estimate');
200 title('Velocity');
201 xlabel('time (s)');
202
203 subplot(313)
204 plot(t,X(3,:)); hold on;
205 plot(t,Xhat(3,:))
206 legend('true', 'estimate');
207 title('Target Acceleration');
208 xlabel('time (s)');
209
210 %plotting state errors
211
212 figure;
213 subplot(311)
214 stairs(t,Ehat(1,:));hold on;
215 plot(t,sig(:,1),'r--',t,-sig(:,1),'r--');
216 title('Position Error');
217 xlabel('time (s)');
218 ylabel('position error');
219
220 subplot(312)
221 stairs(t,Ehat(2,:)); hold on;
222 plot(t,sig(:,2),'r--',t,-sig(:,2),'r--');
223 title('Velocity Error');
224 xlabel('time (s)');
225 ylabel('velocity error');
226
227 subplot(313)
228 stairs(t,Ehat(3,:)); hold on;
229 plot(t,sig(:,3),'r--',t,-sig(:,3),'r--');
```

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