MISSILE STATE ESTIMATION

March 17, 2017

Zhiyuan Cao
304397496
MAE 271B Stochastic Estimation

Contents

Abstract

In this report, first the problem of estimating the relative state of a missile intercept will be introduced, with main objective as minimizing the miss distance. The problem is solved by designing a *Continuous-Time Kalman Filter* and through implementation in MATLAB environment. This is followed by running a Monte Carlo simulation to show the actual error variance matches the *a priori* error variance used in computing Kalman filter gain. Finally, the Gauss-Markov process is replaced with a random telegraph signal to ensure that the Kalman filter is implemented correctly by running, again, a Monte Carlo analysis.

Introduction

Consider the following missile intercept problem,

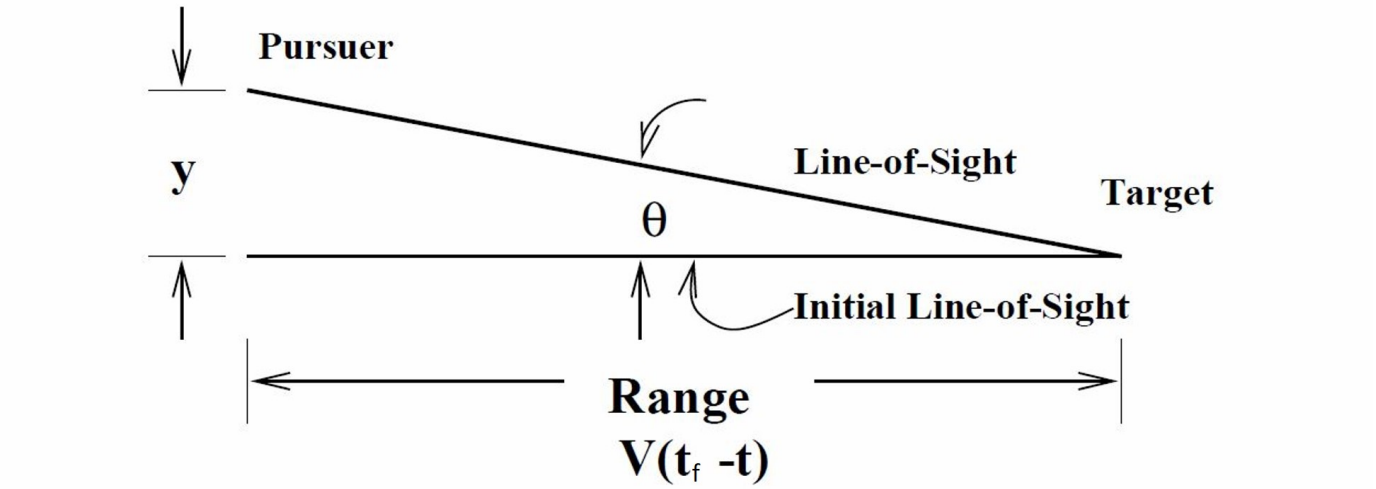

Figure 1: A Illustration of the Missile Intercept [1]

At starting time $t = 0$, the pursuer captures its target and start pursuing, i.e., narrowing the range. At terminal time $t = t_f$, it catches up with the target with horizontal distance $x = V(t_f - t) = 0$. To ensure the success of missile intercept, we need to minimize the vertical distance y at time t_f. The dynamics of this 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 to be zero. The input a_T is the target acceleration and is treated as a random forcing term with an exponential correlation,

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

where the scalar τ is the correlation time. The initial states are given as follows.

$$\begin{aligned} &E[y(t_0)] = 0, && E[v(t_0)] = 0, && \\ &E[y(t_0)^2] = 0, && E[y(t_0)v(t_0)] = 0, && E[v(t_0)^2] = \text{given}. \end{aligned}$$

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

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

Also, it it assumed that z is corrupted by a fading and scintillation noise so that

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

Continuous-Time Kalman Filter

The state-space equation for the missile problem is

$$\begin{aligned} \underbrace{\begin{bmatrix} \dot{y} \\ \dot{v} \\ \dot{a_T} \end{bmatrix}}_{\dot{x}} &= \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & -\frac{1}{\tau} \end{bmatrix}}_{F} \underbrace{\begin{bmatrix} y \\ v \\ a \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}}_{B} a_p + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{G} \omega_{a_T} \\ z &= \underbrace{\begin{bmatrix} \frac{1}{V_c(t_f - t)} & 0 & 0 \end{bmatrix}}_{H} \begin{bmatrix} y \\ v \\ a_T \end{bmatrix} + n \end{aligned} \tag{2}$$

Therefore, the Kalman filter has the form

$$\begin{aligned} \dot{\hat{y}} &= \hat{v} + K_1(z - \frac{\hat{y}}{V_c(t_f - t)}), \\ \dot{\hat{v}} &= -\hat{a_T} + K_2(z - \frac{\hat{y}}{V_c(t_f - t)}) + a_p, \\ \dot{\hat{a_T}} &= -\frac{\hat{a_T}}{\tau} + K_3(z - \frac{\hat{y}}{V_c(t_f - t)}), \end{aligned}$$

where the gains are

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

where p_{ij} are the (i, j) elements of the error covariance matrix is propagated by the *Riccati* Equation

$$\dot{P} = FP + PF^T - PMP + W, \tag{3}$$

where

$$M = \frac{1}{V_c^2 R_1 (t_f - t)^2 + V_c^2 R_2} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad W = GE[a_T^2]G^T = E[a_T^2] \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

From here we can develop the algorithm that solves this problem.

Algorithm 1: Kalman Filter Implementation

Result: Estimate $\hat{x}$, error e
initialize drive ω_{a_T}, noise n;
while $t < t_f$ **do**
  update variance: $P_{k+1} = P_k + \dot{P}_k dt$;
  update Kalman gain: $K_{k+1} = P_{k+1} H_{k+1} / V_{k+1}$;
  update actual state: $x_{k+1} = x_k + \dot{x}_k dt$, $z_{k+1} = H_{k+1} x_{k+1} + n_{k+1}$;
  update estimate state: $\hat{x}_{k+1} = \hat{x}_k + \dot{\hat{x}}_k dt$
end

Implementation

Consider the case where the parameters of the problem are

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

The initial covariance is

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

In order to actually solve such problem, we make several approximations shown as follows. First, we discretize the time t with $\Delta t = 0.01$. Second, we approximate a_T with the following stationary process

$$\begin{aligned} \dot{a_T} &= -\frac{2}{\Delta t} a_T + \omega = 0 \\ \Rightarrow \omega &= \frac{2a_T}{\Delta t} \\ \Rightarrow E[\omega(t)\omega(t)^T] &= \frac{W}{\Delta t} \end{aligned}$$

Based on the model developed and approximations discussed, we realized Kalman filter under Matlab environment. And the results are as follows.

Single Realization

Figure 2 and 3 show the Kalman filter gain and the error variance of the error. As expected, both the Kalman filter gain and the *a priori* variance converge. At the early stage where the error variances are relatively large, the Kalman filter gain reached its peak. As t proceeds to t_f, the coefficient of matrix M approaches to a constant and therefore P is almost in stead-state. The Kalman gain also reflects this property, as it approaches 0 swiftly after the peak.

Figure 3-5 present a comparison between actual state and estimate. As shown here, estimate of position is better than estimate of velocity and acceleration. Through observation, we can also see a delay in estimate in respond to change of actual state in both Figure 4 and 5.

Figure 6 presents the cross-correlation of the residual process. Define residual as follows.

$$\begin{aligned} r &= z - H\hat{x}, \\ \Rightarrow E[r(t)r(\tau)] &= R\delta(t-\tau), \end{aligned}$$

where R is the power spectual density of the residual process. This defines a white-noise process. Through inspection of Figure, we can see that the residual is indeed a white-noise process.

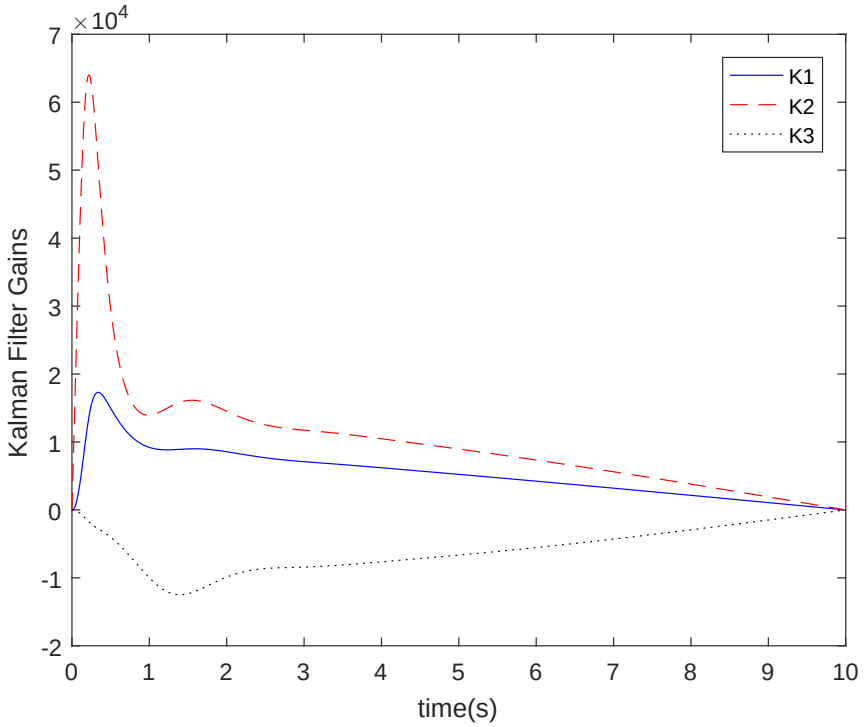

Figure 2: *Filter Gain History*

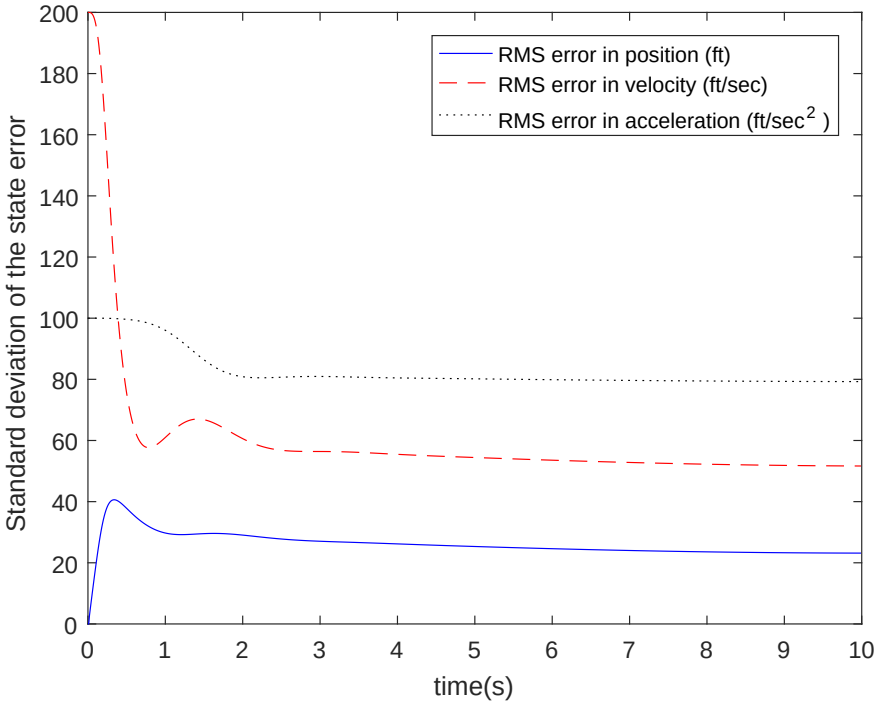

Figure 3: *Evolution of the Estimation Error RMS*

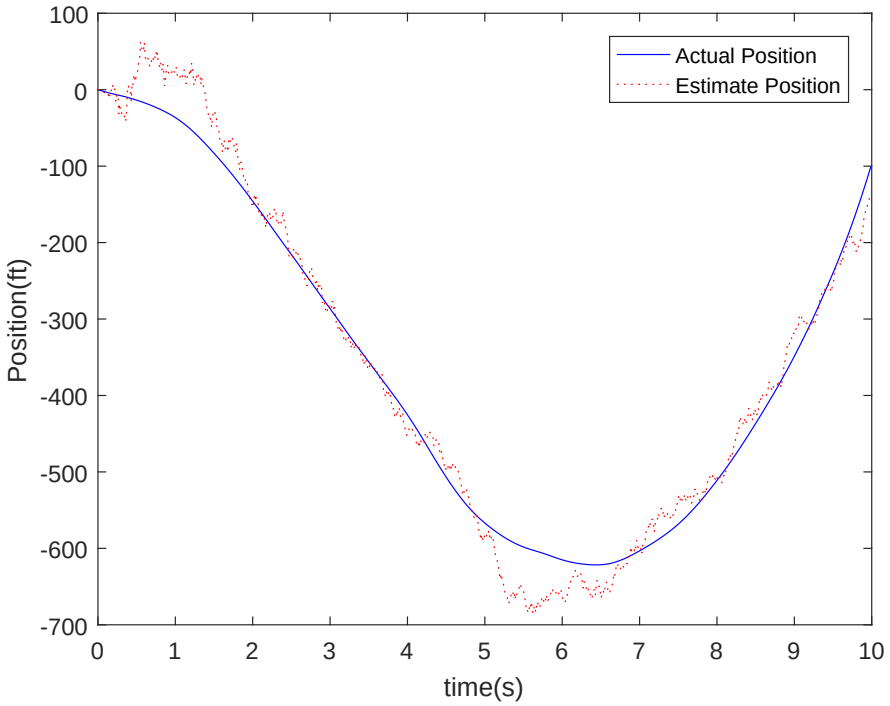

Figure 4: *Actual Position v.s. Estimate Position*

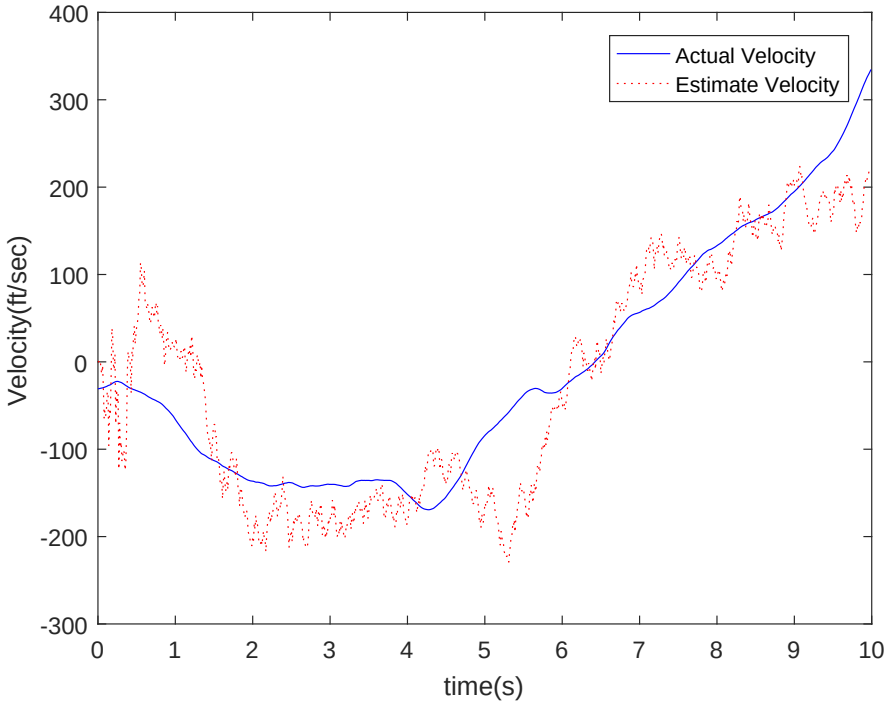

Figure 5: *Actual Velocity v.s. Estimate Velocity*

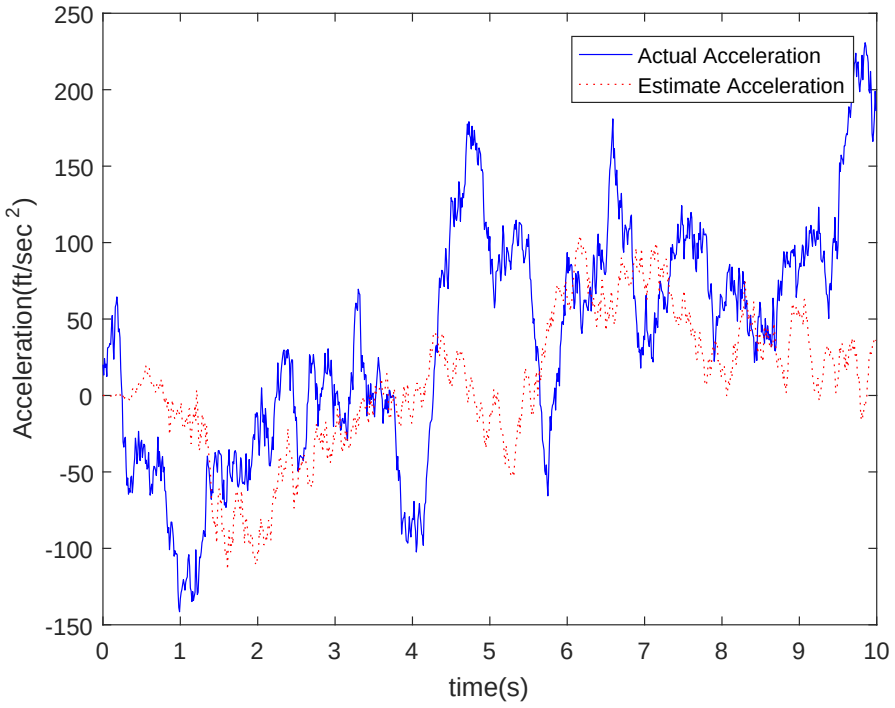

Figure 6: *Actual Acceleration v.s. Estimate Accleration*

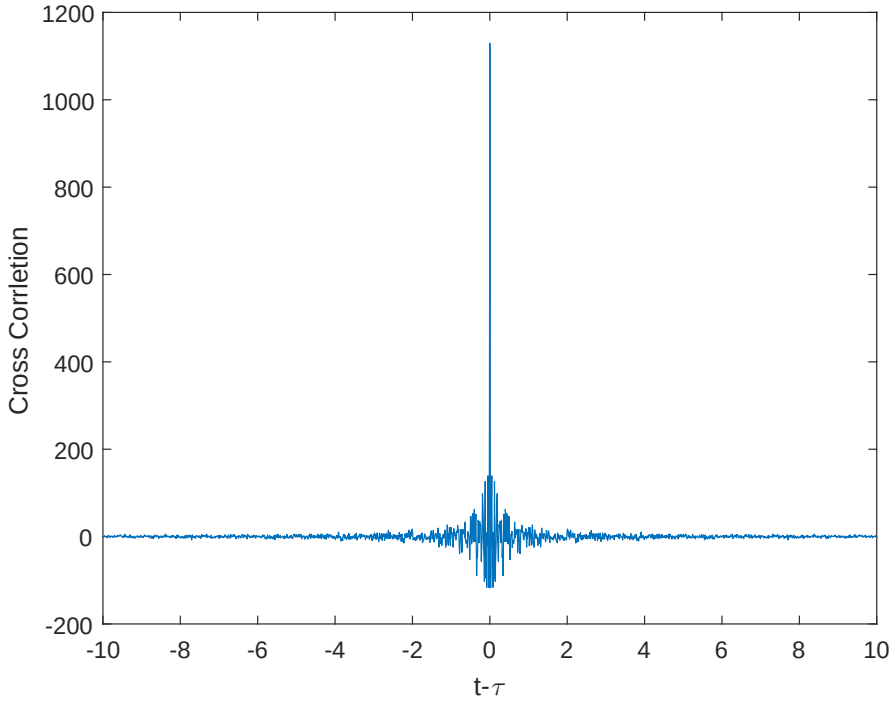

Figure 7: *Cross Correlation of the Residue Process*

Monte Carlo Analysis

For the second part of the implementation, we show by a Monte Carlo simulation, the actual error variance matches the *a priori* variance with 3 different sizes $N = 10,\ 100,\ 1000$ respectively. The error is defined as follows.

$$e = x - \hat{x}$$

Figure 8-10 presents the ensemble average of the position, velocity and acceleration marked in red full line (due to lack of space, no legend is shown), bracketed by one-sigma bound of the *a priori* variance. The ensemble average is calculated as follows.

$$e^{ave}(t_i) = \frac{1}{N}\sum_{j=1}^{N} e^j(t_i).$$

Figure 11-14 shows a comparison between actual error variance and *a priori* variance. As the number of realization increases, difference between the actual variance calculated and *a priori* variance shrinks. The variance is calculated as follows.

$$P^{ave}(t_i) = \frac{1}{N-1}\sum_{j=1}^{N}[e^j(t_i) - e^{ave}(t_i)][e^j(t_i) - e^{ave}(t_i)]^T$$

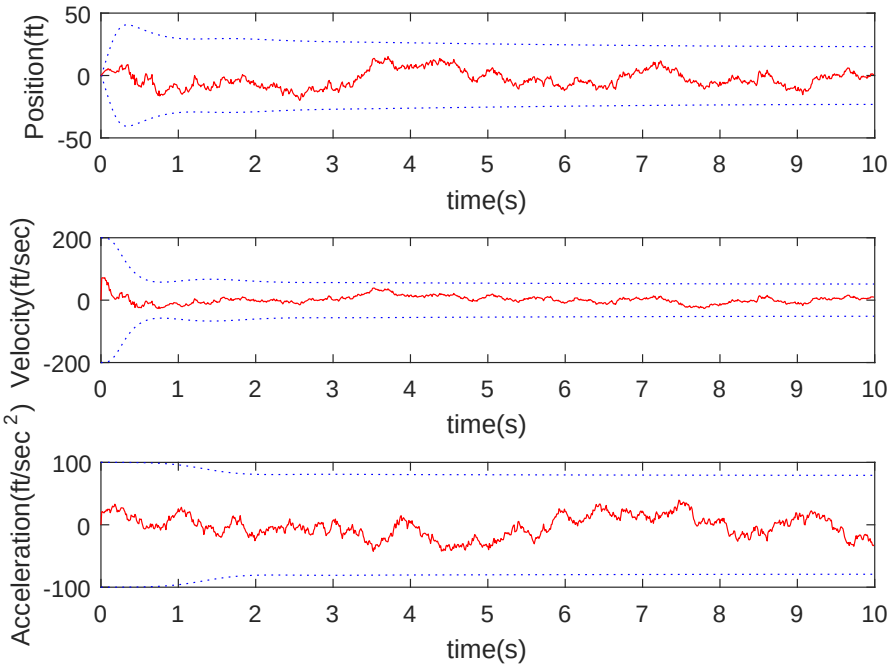

Figure 8: *Ensemble Average of the states with N = 10*

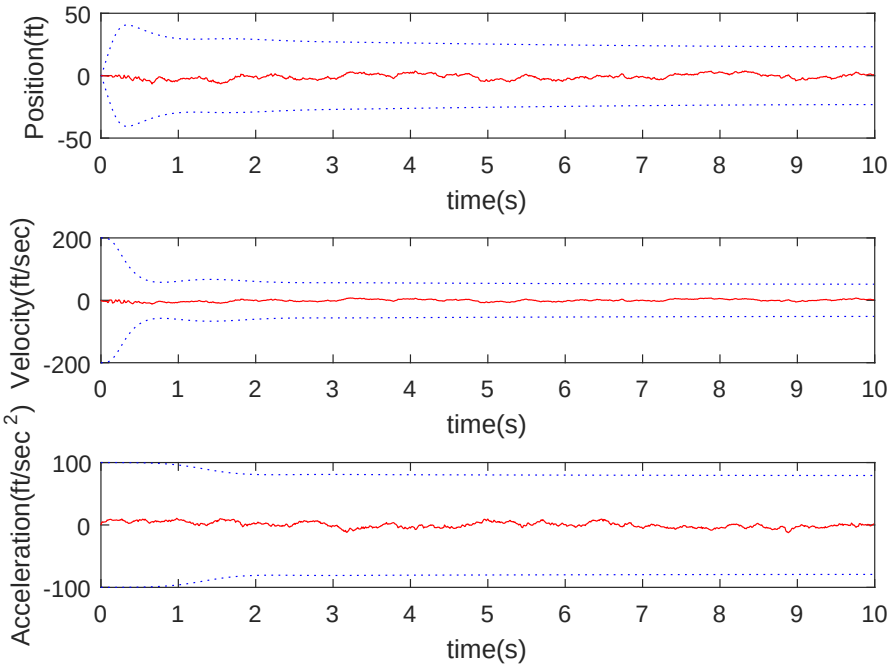

Figure 9: *Ensemble Average of the states with $N = 100$*

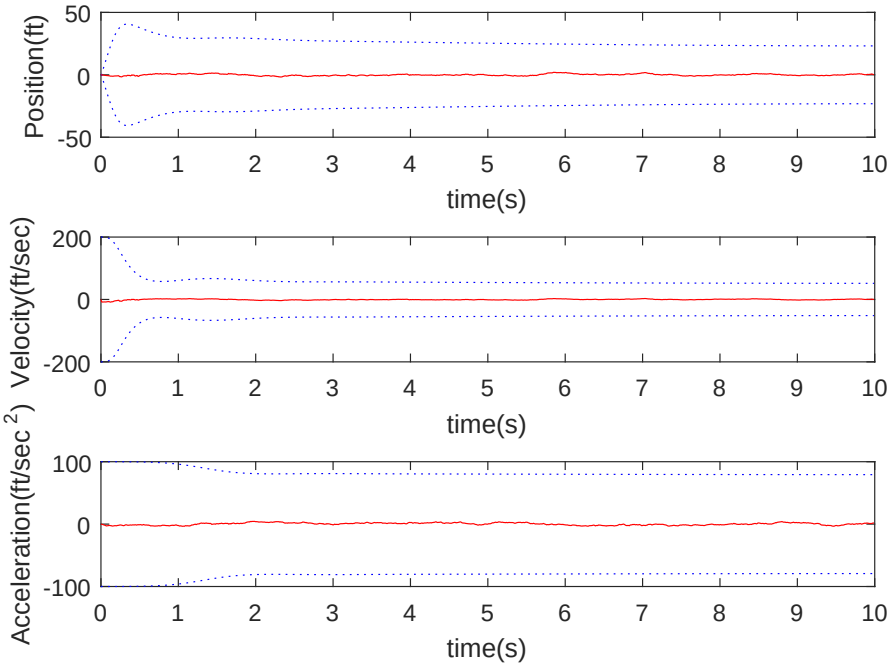

Figure 10: *Ensemble Average of the states with $N = 1000$*

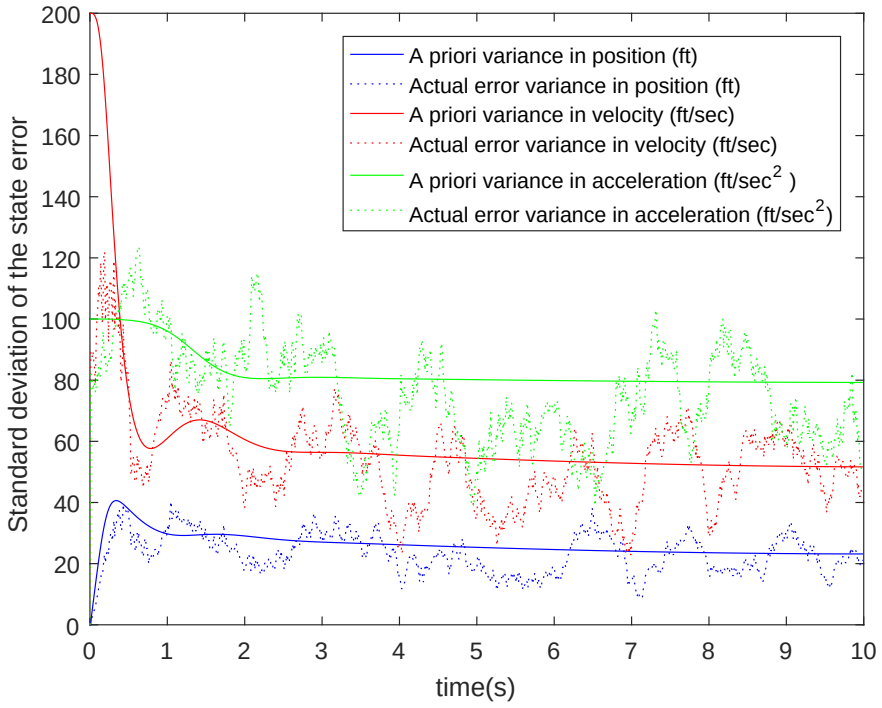

Figure 11: *Actual Error Variance v.s. A Priori Variance with N = 10*

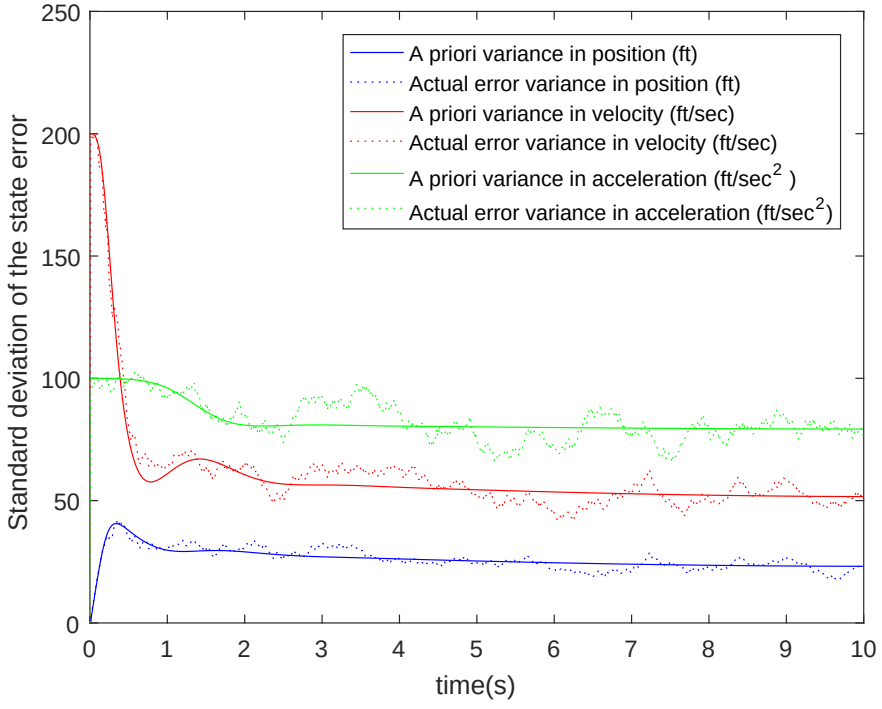

Figure 12: *Actual Error Variance v.s. A Priori Variance with N = 100*

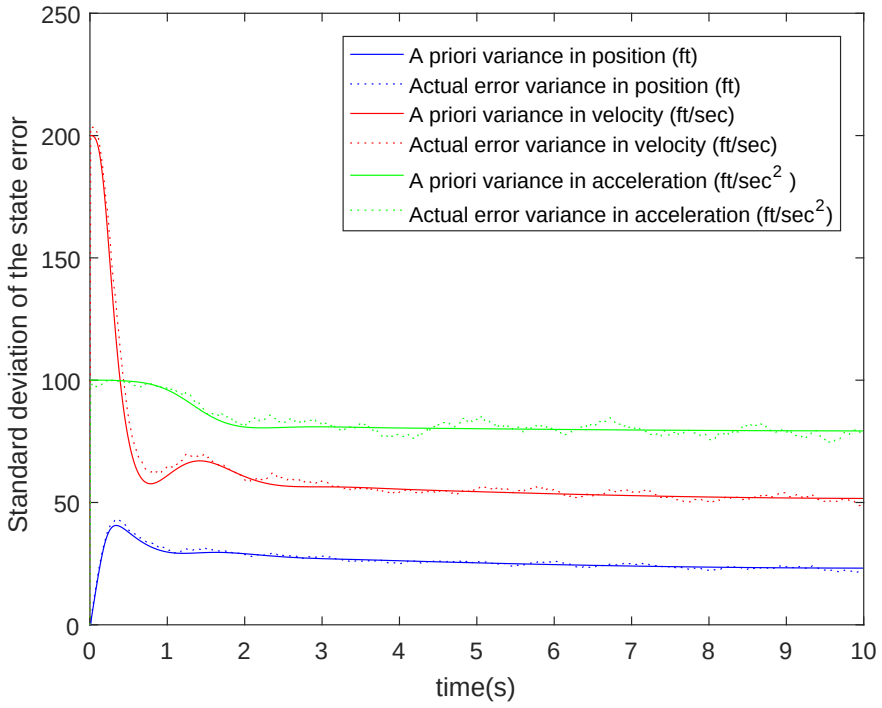

Figure 13: *Actual Error Variance v.s. A Priori Variance with N = 1000*

Random Telegraph Signal Model

Recall that we made a few approximation when implementing the Kalman filter. Now we utilize a random telegraph signal, or *Burst Noise*, to verify that the Kalman filter has been implemented corrected this way. In this model, the value a_T changes sign at random times given by a *Poisson* probability, defined as follows.

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

where U is the output of $[0, 1]$ uniform distribution and $\lambda = 0.25$. A single realization of a_T is shown in Figure 14.

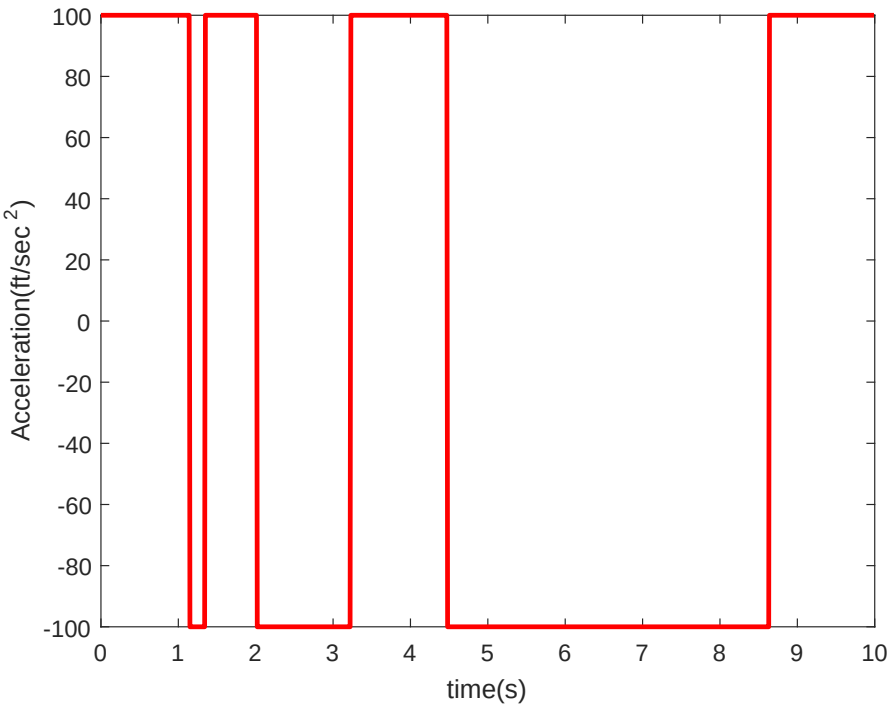

Figure 14: *Single Realization of Acceleration Input using Random Telegraph Signal Model*

Essentially, we replaced the system input generated by a *Gauss-Markov* process with a model described above and run Monte Carlo Simulation based on such input. Again, we have $N = 10,\ 100,\ 1000$ respectively. The results are shwon in Figure 15-20. As shown in Figure 15-17, as sample size N increases, then ensemble average converges to zero, and in Figure 18-20, the actual error variance converge to *a priori* variance.

One interesting observation made during the implementation of the random telegraph signal is that the initial acceleration, instead of be selected randomly between a_0 and $-a_0$, is set to a fixed value and therefore the Monte Carlo analysis results in a difference (less) in ensemble mean and variance in early stage, i.e, when t is small. It converges to zero at steady state. At the beginning, it was believed to be some property of the random telegraph signal. However, this was a pure implementation error, and such less in statistics make sense since the beginning of the stage is deterministic and therefore there is less variance.

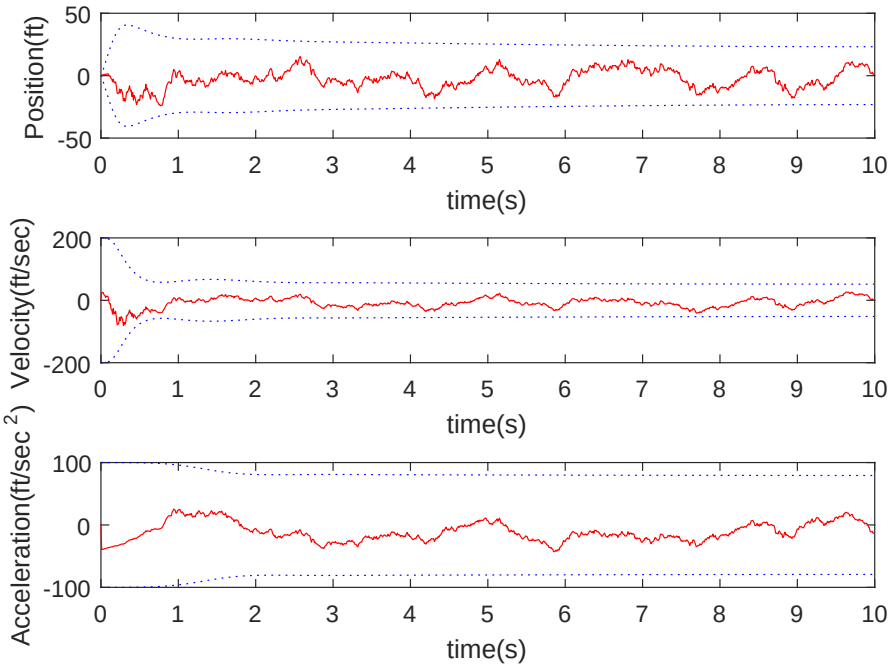

Figure 15: *Ensemble Average of the states with $N = 10$*

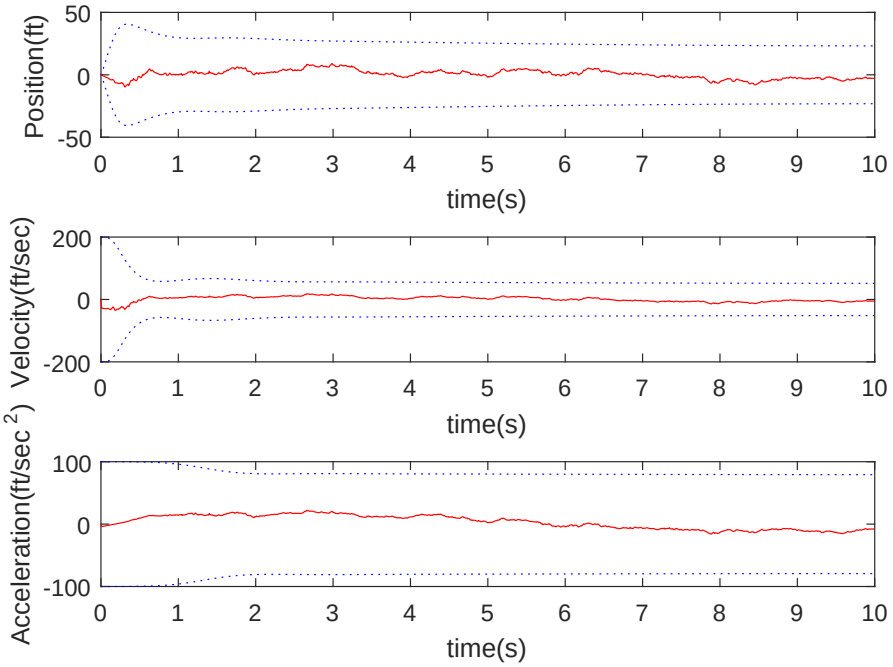

Figure 16: *Ensemble Average of the states with $N = 100$*

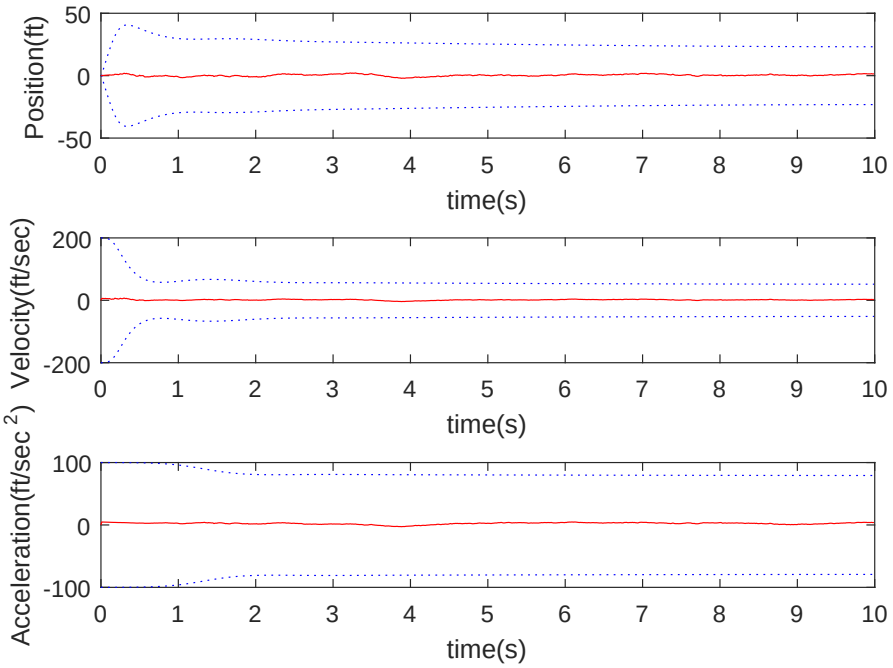

Figure 17: *Ensemble Average of the states with $N = 1000$*

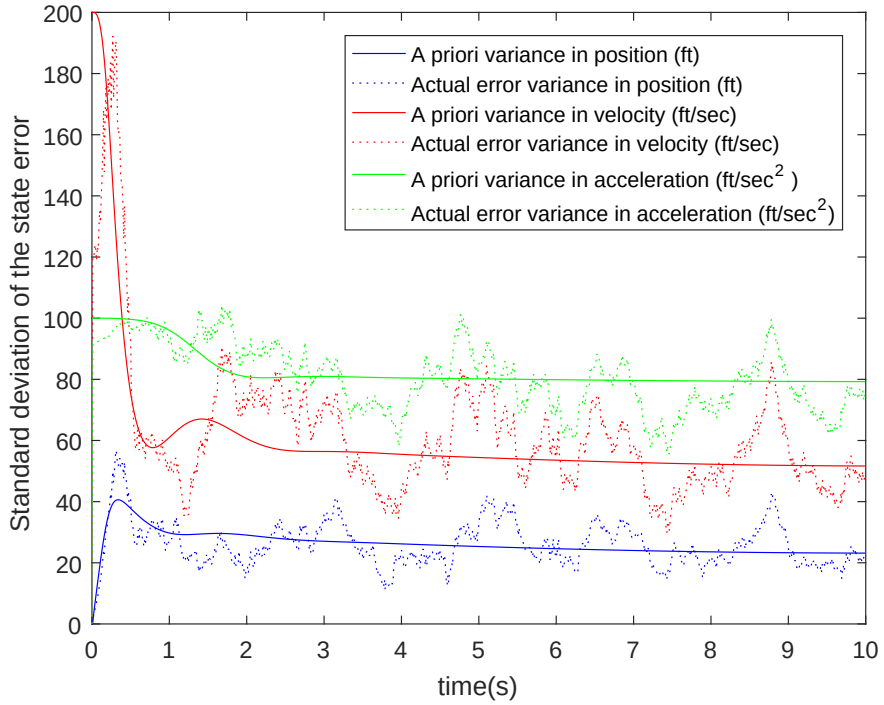

Figure 18: *Actual Error Variance v.s. A Priori Variance with $N = 10$*

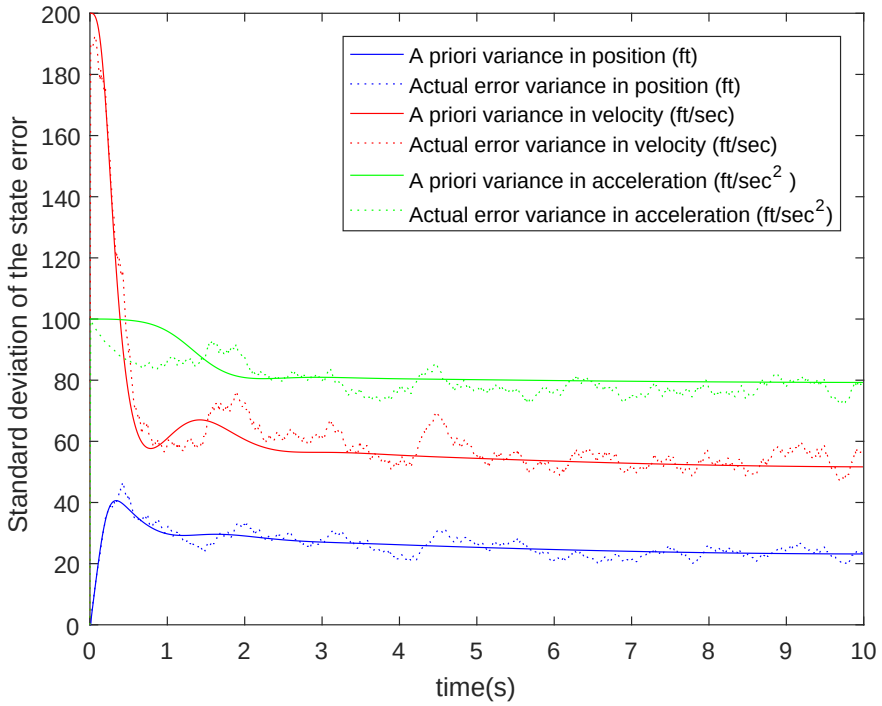

Figure 19: *Actual Error Variance v.s. A Priori Variance with N = 100*

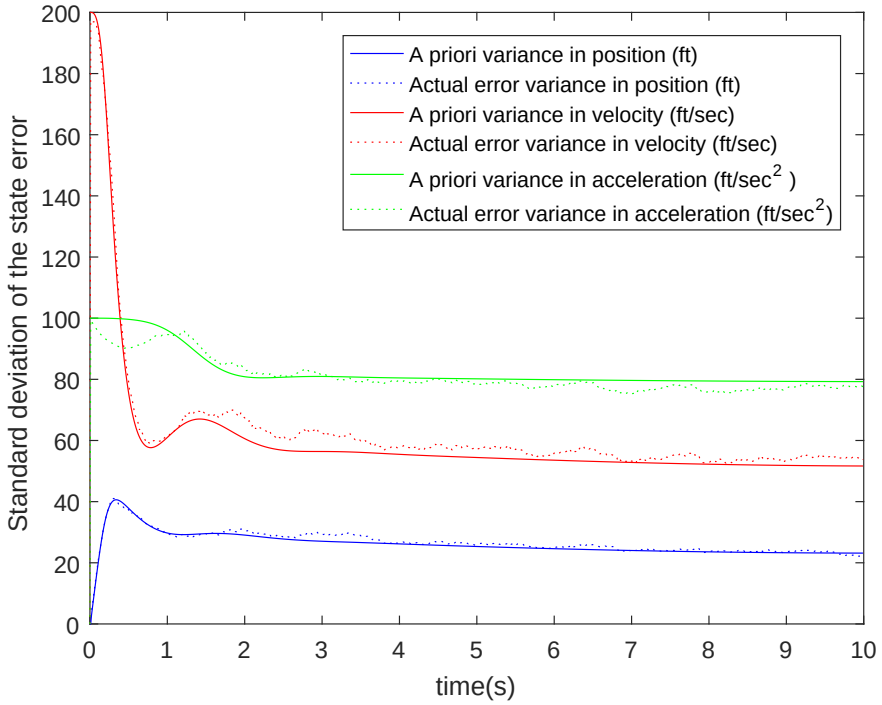

Figure 20: *Actual Error Variance v.s. A Priori Variance with N = 1000*

Conclusion

In this report, the missile intercept problem is first introduced, followed by formation of the *continuous-time Kalman filter*. Based on the model, an implementation under Matlab environment is carried out and the results are presented and discussed. Based on the result of Monte Carlo simulation of both the original model and the random telegraph signal model, the implementation of Kalman filter can be concluded as successful.

During the process of implementation several mistakes were made and fixed, which provided some insights to the mechanism of Kalman filter. For instance, the initial state being deterministic instead of random reduces the actual error variance since it is less "random" and it does not affect later stage due to it is uncorrelated.

In the future, the mechanism of random telegraph signal can be further explored and its implementation in different fields can be further explored. Aside from that, the continuous Kalman filter, althrough being continuous, is implemented through discretization of t, which bears some level of similarity to discrete time Kalman filter, and therefore a comparison between two filters can be made and discussed.

Appendix

Here the main Matlab Kalman filter code is attached as well as Random Telegraph Signal realization.

Kalman Filter Main

```
% ------------------------------------------------
% MAE 271B Project
% Zhiyuan Cao
% 02/27/17
% ------------------------------------------------
clc;clear;
% ------------------------------------------------
% GLOBAL CONSTANTS
% ------------------------------------------------
vc = 300;                          % ft/sec
tf = 10;                           % sec
R1 = 15e-6;                        % rad^2sec
R2 = 1.67e-3;                      % rad^2sec^3
tau = 2;                           % sec
dt = 0.01;                         % sec
sdt = sqrt(dt);                    % sqrt(dt)
t = linspace(0, tf-dt, tf/dt)';  % discrete t
m = length(t);
ns = 1000;                           % number of simulation
% ------------------------------------------------
```

```
% STATS
% ------------------------------------------------
y_mea = 0;
v_mea = 0;
a_mea = 0;
n_mea =0;
y_sig = 0;
v_sig = 200;
v_var = 200^2;
a_sig = 100;
a_var = 100^2;
V = zeros(1, m);                    % State Noise PSD
for ii = 1 : m
    V(ii) = R1 + R2/((tf-t(ii))^2);
end
% ------------------------------------------------
% STATE SPACE
% ------------------------------------------------
F = [0 1 0; 0 0 -1; 0 0 -1/tau];
B = [0 1 0]';
G = [0 0 1]';
W = G*a_sig^2*G';
H = zeros(3, m);                   % Measurement Matrix
M = zeros(3, 3, m);
for ii = 1 : m
    H(:, ii) = [1/(vc*(tf-t(ii))) 0 0];
    M(:, :, ii) = H(:, ii)*H(:, ii)'/V(ii);
end
% ------------------------------------------------
% GLOBAL DATA STRUCTURE
% ------------------------------------------------
err_g = zeros(3, m, ns);           % Global Err
err_aver = zeros(3, m);            % Global Err Aver
P_aver = zeros(3, 3, m);           % Global Var Aver
res_g = zeros(1, m, ns);           % Global Res
res_aver = zeros(1, m);            % Global Res Aver
% ------------------------------------------------
% Main Loop
% ------------------------------------------------
for jj = 1:ns
    % --------------------------------------------
    % DATA STRUCTURE
    % --------------------------------------------
    Wat = zeros(1, m);           % Process Noise
    x = zeros(3, m);             % Actual State x
```

```
dx = zeros (3, m);          % Actual Delta x
n = zeros(1, m);            % State Noise
z = zeros(1, m);            % Measurement
xhat = zeros(3, m);         % Estimate State x
dxh = zeros(3, m);          % Estimate Delta x
P = zeros(3, 3, m);         % Variance
K = zeros(3, m);            % Kalman Gain
err = zeros(3, m);          % Error
res = zeros(1, m);          % Residues
% ------------------------------------------------
% INITILIZE VALUES
% ------------------------------------------------
v0 = normrnd(v_mea, v_sig);
a0 = normrnd(a_mea, a_sig);
Wat(1) = normrnd(a_mea, a_sig/sdt);
x(:, 1) = [0, v0, a0]';
dx(:, 1) = F*x(:, 1) + G*Wat(1);
n(1) = normrnd(0, sqrt(V(1)/dt));
z(:, 1) = H(:, 1)'*x(:, 1) + n(1);
P(:, :, 1) = [0 0 0; 0 v_var 0; 0 0 a_var];
K(:, 1) = P(:, :, 1)*H(:, 1)/V(1);
% ------------------------------------------------
% UPDATE
% ------------------------------------------------
for ii = 1 : m - 1
    % UPDATE VARIANCE
    dp = F*P(:, :, ii) + P(:, :, ii)*F'...
        - P(:, :, ii)*M(:, :, ii)*P(:, :, ii)...
        + W;
    P(:, :, ii + 1) = P(:, :, ii) + dp*dt;
    % UPDATE KALMAN GAIN
    K(:, ii + 1) = P(:, :, ii + 1)*H(:, ii + 1)/V(ii + 1); % ii+1?
    % UPDATE INPUT
    Wat(ii + 1) = normrnd(a_mea, sqrt(a_var/dt));
    n(ii + 1) = normrnd(n_mea, sqrt(V(ii)/dt));
    % UPDATE ACTUAL STATE
    dx(:, ii + 1) = F*x(:, ii) + G*Wat(ii + 1);
    x(:, ii + 1) = x(:, ii) + dx(:, ii + 1)*dt;
    z(:, ii + 1) = H(:, ii + 1)'*x(:, ii + 1) + n(ii + 1);
    % UPDATE ESTIMATE STATE
    dxh(:, ii + 1) = F*xhat(:, ii) + K(:, ii)...
        *(z(:, ii) - H(:, ii)'*xhat(:, ii));
    xhat(:, ii + 1) = xhat(:, ii) + dxh(:, ii + 1)*dt;
    err(:, ii + 1) = xhat(:, ii + 1) - x(:, ii + 1);
    % UPDATE RESIDUE
```

```
        res(ii + 1) = z(:, ii + 1) - H(:, ii + 1)'*xhat(:, ii + 1);
    end
    err_g(:, :, jj) = err;
    res_g(:, :, jj) = res;
end
% ------------------------------------------------
% DATA ANALYSIS
% ------------------------------------------------
for jj = 1 : ns
   err_aver = err_g(:, :, jj) + err_aver;
   res_aver = res_g(:, :, jj) + res_aver;
end
err_aver = err_aver/ns;
res_aver = res_aver/ns;
for jj = 1 : ns
    for ii = 1 : m
        P_aver(:, :, ii) = P_aver(:, :, ii) + (err_g(:, ii, jj) - err_aver(:, ii))...
            *(err_g(:, ii, jj) - err_aver(:, ii))';
    end
end
res_cor = xcorr(res_aver);
P_aver = P_aver/ns;
%P_aver = P_aver/ns;
% ------------------------------------------------
% PLOTS
% ------------------------------------------------
% figure (1), plot(t, K(1, :), 'b'),
% hold on,    plot(t, K(2, :), 'r--'),
% hold on,    plot(t, K(3, :), 'k:'),
% hold off,
% xlabel('time(s)') % x-axis label
% ylabel('Kalman Filter Gains') % y-axis label
% legend('K1', 'K2', 'K3');
%
% figure (2), plot(t, sqrt(squeeze(P(1, 1, :))), 'b'),
% hold on,    plot(t, sqrt(squeeze(P(2, 2, :))), 'r--'),
% hold on,    plot(t, sqrt(squeeze(P(3, 3, :))), 'k:'),
% hold off,
% xlabel('time(s)') % x-axis label
% ylabel('Standard deviation of the state error') % y-axis label
% legend('RMS error in position (ft)', ...
%     'RMS error in velocity (ft/sec)', ...
%     'RMS error in acceleration (ft/sec^2 )');
%
% figure (3), plot(t, x(1, :), 'b-'),
```

```
% hold on,    plot(t, xhat(1,:), 'r:')
% hold off,
% xlabel('time(s)') % x-axis label
% ylabel('Position(ft)'), % y-axis label
% legend('Actual Position', 'Estimate Position');
%
% figure (4), plot(t, x(2, :), 'b-');
% hold on,    plot(t, xhat(2,:), 'r:')
% hold off,
% xlabel('time(s)') % x-axis label
% ylabel('Velocity(ft/sec)'), % y-axis label
% legend('Actual Velocity', 'Estimate Velocity');
%
% figure (5), plot(t, x(3, :), 'b-'),
% hold on,    plot(t, xhat(3,:), 'r:')
% hold off,
% xlabel('time(s)') % x-axis label
% ylabel('Acceleration(ft/sec^2)'), % y-axis label
% legend('Actual Acceleration', 'Estimate Acceleration');

figure (6),
subplot(3,1,1), plot(t, sqrt(squeeze(P(1, 1, :))), 'b:'),
hold on,        plot(t, -sqrt(squeeze(P(1, 1, :))), 'b:'),
hold on,        plot(t, err_aver(1, :), 'r'),
xlabel('time(s)') % x-axis label
ylabel('Position(ft)'), % y-axis label
hold off;
subplot(3,1,2), plot(t, sqrt(squeeze(P(2, 2, :))), 'b:'),
hold on,        plot(t, -sqrt(squeeze(P(2, 2, :))), 'b:'),
hold on,        plot(t, err_aver(2, :), 'r'),
xlabel('time(s)') % x-axis label
ylabel('Velocity(ft/sec)'), % y-axis label
hold off;
subplot(3,1,3), plot(t, sqrt(squeeze(P(3, 3, :))), 'b:'),
hold on,        plot(t, -sqrt(squeeze(P(3, 3, :))), 'b:'),
hold on,        plot(t, err_aver(3, :), 'r'),
xlabel('time(s)') % x-axis label
ylabel('Acceleration(ft/sec^2)'), % y-axis label
hold off;

figure (7), plot(t, sqrt(squeeze(P(1, 1, :))), 'b'),
hold on,    plot(t, sqrt(squeeze(P_aver(1, 1, :))), 'b:'),
hold on,    plot(t, sqrt(squeeze(P(2, 2, :))), 'r'),
hold on,    plot(t, sqrt(squeeze(P_aver(2, 2, :))), 'r:'),
hold on,    plot(t, sqrt(squeeze(P(3, 3, :))), 'g'),
```

```
hold on,    plot(t, sqrt(squeeze(P_aver(3, 3, :))), 'g:'),
hold off,
xlabel('time(s)') % x-axis label
ylabel('Standard deviation of the state error') % y-axis label
legend('A priori variance in position (ft)', ...
    'Actual error variance in position (ft)',...
    'A priori variance in velocity (ft/sec)', ...
    'Actual error variance in velocity (ft/sec)',...
    'A priori variance in acceleration (ft/sec^2 )', ...
    'Actual error variance in acceleration (ft/sec^2)');

% c = linspace(-10,10,1999);
% figure (8), plot(c,res_cor),
% xlabel('t-\tau') % x-axis label
% ylabel('Cross Corrletion'); % y-axis label
```

Random Telegraph Signal Realization

```
function [a0, acc] = acc_gen(tf, dt, at)
lam = 0.25; tc = 0;
t = linspace(0, tf-dt, tf/dt)';
m = length(t);
acc = zeros(1, m);
if (round(rand) == 1)
    a0 = at;
else
    a0 = -at;
end
ii = 1;
while (tc <= tf)
   tc(ii + 1) = tc(ii) - log(rand)/lam;
   ii = ii + 1;
end
ii = 2;
for jj = 1:length(t)
    if(t(jj) > tc(ii))
        at = -at;
        ii = ii + 1;
    end
    acc(jj) = at;
end
% plot(t, acc, 'r', 'LineWidth',2),
% xlabel('time(s)'),
% ylabel('Acceleration(ft/sec^2)');
```

```
end
```

References

[1] Speyer, J. L., & Chung, W. H. *Stochastic processes, estimation, and control.* Society for Industrial and Applied Mathematics, 2008.