MAE 271B Project
Missile State Estimation

Aruul Mozhi Varman Senthilkumar

aruul@ucla.edu

Abstract

The main goal of the project is to estimate the missile's lateral position, velocity and acceleration using the line-of-sight angle measurements from the radar. A Kalman-Bucy filter is constructed for a Gauss-Markov process, and subsequently the same filter is employed to track the missile with input modeled using a more realistic random telegraph signal characterised by a similar correlation function.

1. Introduction

A pursuer with radar launches a missile to intercept the target and the goal is to estimate the state of the missile as it approaches the target. The problem is formulated in a 2-dimensional world, accordingly the pursuer has a 2-dimensional radar and the objective is to track the lateral position, velocity and acceleration of the missile.

The measurement z from the radar is corrupted by fading and scintillation noise n and is characterized as

$$z = \theta + n$$

Here, θ is the line-of-sight angle to the target. For $|\theta| << 1$, it is approximated as

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

where $V_c = 300\, ft\, s^{-1}$ and $t_f = 10\, s$ is the terminal time. The statistics of the measurement process is as follows

$$E[n(t)] = 0, E[n(t)n(\tau)] = V\delta(t - \tau) = [R_1 + \frac{R_2}{(t_f - t)^2}]\delta(t - \tau)$$

Here, $R_1 = 15 \times 10^{-6}$ and $R_2 = 1.67 \times 10^{-3}\, rad^2\, s^3$. δ is the Dirac delta function and $\tau = 2$ is the correlation time variable.

The dynamics of the missile are

$$\dot{y} = v, \dot{v} = a_p - a_T;$$

Here, y and v are lateral position and velocity of the missile. a_p is the missile acceleration and it is zero in this case. a_T is the input to the system and it is modeled on the target acceleration as a random forcing function with exponential correlation. The auto-correlation function of a_T is given by eq. (1) and the statistics are as follows:

$$E[a_T] = 0, [a_T^2] = (100ft\,s^{-2})^2$$

$$E[a_T(t)a_T(s)] = E[a_T^2]e^{\frac{-|t-s|}{\tau}} \tag{1}$$

The initial lateral position y is zero and the initial lateral velocity v is assumed to be normally distributed with the following statistics to capture the launching error:

$$E[y(t_0)] = 0, E[v(t_0)] = 0, E[y(t_0)^2] = 0, E[y(t_0)v(t_0)] = 0, E[v(t_0)^2] = (200ft\,s^{-1})^2$$

2. Theory

The estimator is based on the Kalman-Bucy filter, a continuous-time linear minimum variance estimator. The filter is an optimal estimator and a conditional mean estimator for Gauss-Markov process. If the additive noise is uncorrelated the filter is equivalent to Kalman Filter in structure.

In continuous-time filter there is no separation between propagation and the measurement update steps and they occur simultaneously as in eq. (2).

$$d\hat{x}(t) = F(t)\hat{x}(t)dt + K(t)(dz(t) - H(t)\hat{x}(t)dt) \tag{2}$$

Here, $F(t)$ is the system matrix and $H(t)$ is the observation model. $dz(t)$ is the measurement at time t and the Kalman gain $K(t)$ given by eq. (3) is used to incorporate the measurement in the estimate.

$$K(t) = P(t)H(t)^T V(t)^{-1} \tag{3}$$

Here, $P(t)$ is the error variance estimate and $V(t)$ is the power spectral density matrix of the additive white noise n in the measurement process.

The dynamics of the process error variance estimate $P(t)$ is determined to be the famous continuous-time algebraic Riccati equation as in the eq. (4).

$$\dot{P}(t) = F(t)P(t) + P(t)F(t)^T - P(t)H(t)^T V(t)^{-1}H(t)P(t) + G(t)W(t)G(t)^T \tag{4}$$

Here, $G(t)$ is the measurement noise model and W is the power spectral density matrix corresponding to the additive process noise w_{a_T}.

3. Algorithm

A state-space model of the dynamics is developed for use in the estimator and the state-space equation is as follows:

$$\dot{x} = Fx + Gw_{a_T}, x = \begin{bmatrix} \dot{y} \\ \dot{v} \\ \dot{a_T} \end{bmatrix}, F = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & -\frac{1}{\tau} \end{bmatrix}, G = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

An additional state a_T based on the input is added to the state-space equations of the missile to track the dynamics of the exponentially correlated input process a_T. This is based on the fact continuous-time Gauss-Markov process evolve exponentially with time and the input in this case target acceleration a_T is modelled as an exponentially correlated process. The state-space model of the measurement process is

$$z = Hx + n, H = \begin{bmatrix} \frac{1}{V_c(t_f - t)} & 0 & 0 \end{bmatrix}$$

In the state-space equations the random variables are denoted by small letters and accordingly the deterministic components are labeled with capital letters.

The initial a priori error covariance is determined from the problem parameters as follows:

$$P(0) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & (200 ft\, s^{-1})^2 & 0 \\ 0 & 0 & (100 ft\, s^{-2})^2 \end{bmatrix}$$

Similarly, the power spectral density matrices for the additive process noise W and the additive non-stationary white measurement noise V are determined to be

$$V = [R_1 + \frac{R_2}{(t_f - t)^2}], W = [(100 ft\, s^{-2})^2]$$

Once, the estimator performs well on the Gauss-Markov process then estimator is used on a more realistic process modeled after the random telegraph signal. In this improved process model, the value of input a_T is $\pm 100 ft\, s^{-2}$ that changes sign at random times given by a Poisson probability. The initial state of the input $a_T(0)$ is assumed to be $\pm a_T$ with a probability of 0.5 and $a_T(t)$ changes polarity at Poisson times. The probability of k sign changes in a time interval of length T, $P(k(T))$ is given by

$$P(k(T)) = \frac{(\lambda T)^k e^{-\lambda T}}{k!}$$

where λ is the rate and is equal to $0.25\, s^{-1}$ for our problem. The mean of the input acceleration a_T based on the random telegraph signal is found to be $E[a_T] = 0$ and the auto-correlation function of a_T for this model is given by eq. (5)

$$R_{a_T a_T}(t, s) = {a_T}^2 e^{-2\lambda |t-s|} \tag{5}$$

In this case $\frac{1}{\tau} = 2\lambda = 0.5\, s^{-1}$, then as we can see the auto-correlation function of the Gauss-Markov process given by equation (1) is same as the random telegraph signal given by eq. (5) where t and s are correlation time variables. Also, the mean of both the process are zero.

The theory for developing estimators for Gauss-Markov process is well understood and once we have an estimator for the Gauss-Markov process the main objective is use to the estimator with a more realistic model of the increment process a_T given by random telegraph signal.

A discretized implementation of the filter with discretization time $dt = 0.01\, s$ is run till the terminal time $t_f = 10\, s$. The power spectral density matrices W and V are divided by the discretization time dt as in eq. (6) to account for the discretization.

$$W_{discrete} = \frac{W}{dt}, V_{discrete} = \frac{V}{dt} \tag{6}$$

In the discretized implementation, the base of δ is made to be dt and decreasing the amplitude of the δ by a factor of dt as in eq. (6) ensures that the area or magnitude of the impulse remains unchanged. Essentially the power of the discretized system is modified so that the energy of the discretized system is same as that of the original continuous-time system.

The switching times for the random telegraph signal is given by Poisson probability and the next switching time can be computed from the previous switching time as in eq. (7).

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

where t_n is the time of n^{th} sign change and t_{n+1} is the time for the following sign change. U is a random variable with uniform density function defined on the range $[0, 1]$. In the discrete time implementation of the estimator the switching times t_n are rounded to the closest valid discrete time intervals.

The results and performance of the estimator on Gauss-Markov process and the increment process produced by random telegraph signal is discussed in the following sections 4 and 5.

4. Results

The estimator is run for 1000 different trial runs for terminal time $t_f = 10\,s$ and the results are presented as figures. The Kalman filter gains $K(t)$, the actual RMS estimation error and the Kalman filter estimated RMS error are presented in Fig. 1 and 2 respectively. The error variance corresponding to the states are presented in 3. The true state values and the filter estimates are presented in Fig. 5, 6 and 7 corresponding to lateral position, velocity and acceleration for Gauss-Markov process. The measurements z corresponding to presented trial is show in Fig. 4. The error in state estimates $e = x - \hat{x}$ corresponding to the run in the previous results are present in Fig. 8, 9 and 10 along side the estimated error standard deviation $\pm 1\sigma$.

In the same way, the performance of the filter for random telegraph signals are illustrated for one of the experimental runs. The Fig. 11 represents the measurements z for the run. The Fig. 12, 13 and 14 compare the true values of the states with the estimated values for random telegraph signal. In addition the error e in the estimates are presented in Fig. 15, 16 and 17 for all the states along with their estimated $\pm 1\sigma$ error standard deviations.

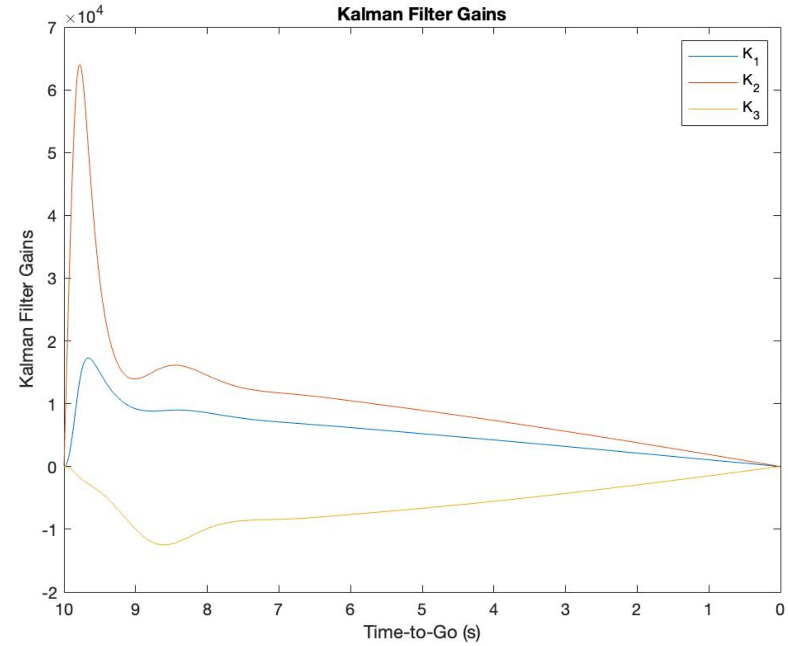

Figure 1: Kalman Filter Gain History

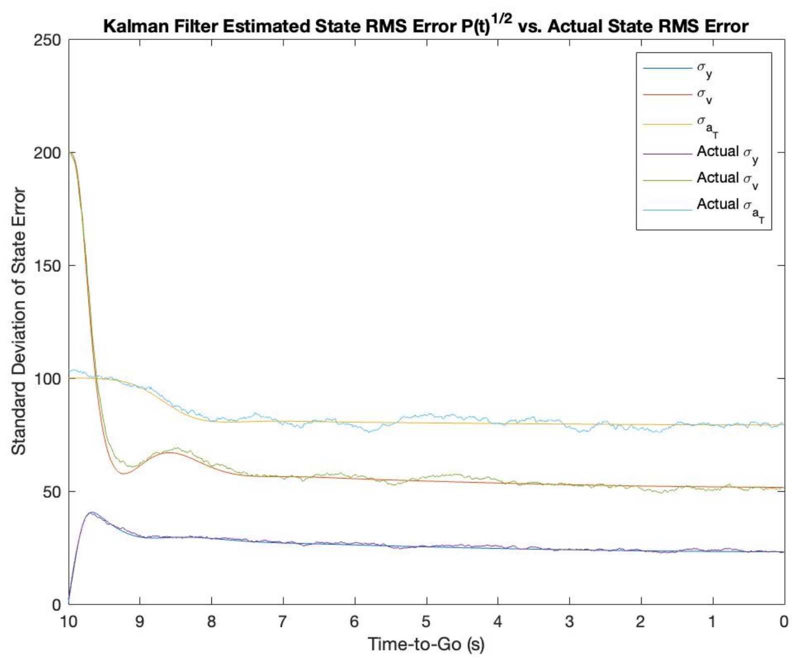

Figure 2: Comparison of Kalman filter estimated state RMS error and actual state RMS error in estimation.

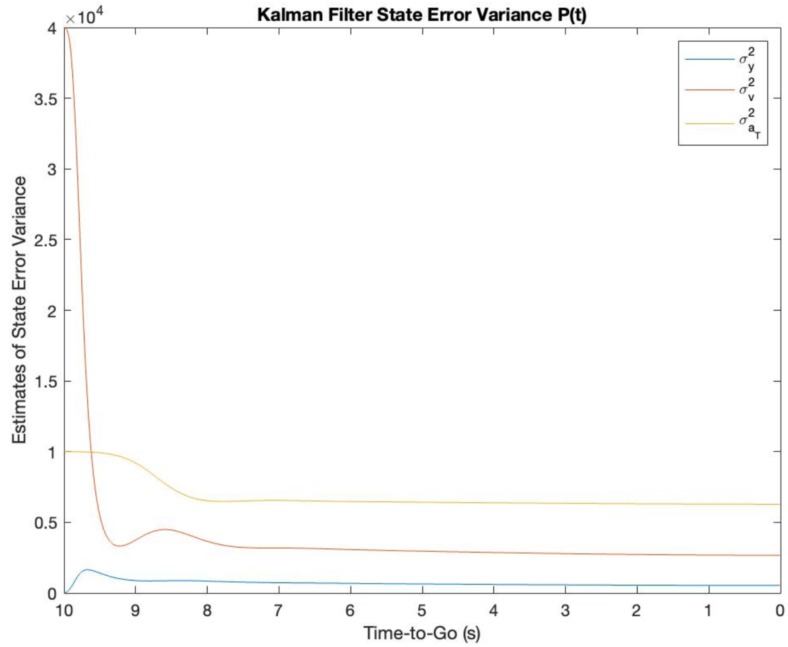

Figure 3: Kalman filter estimated state error variance $P(t)$

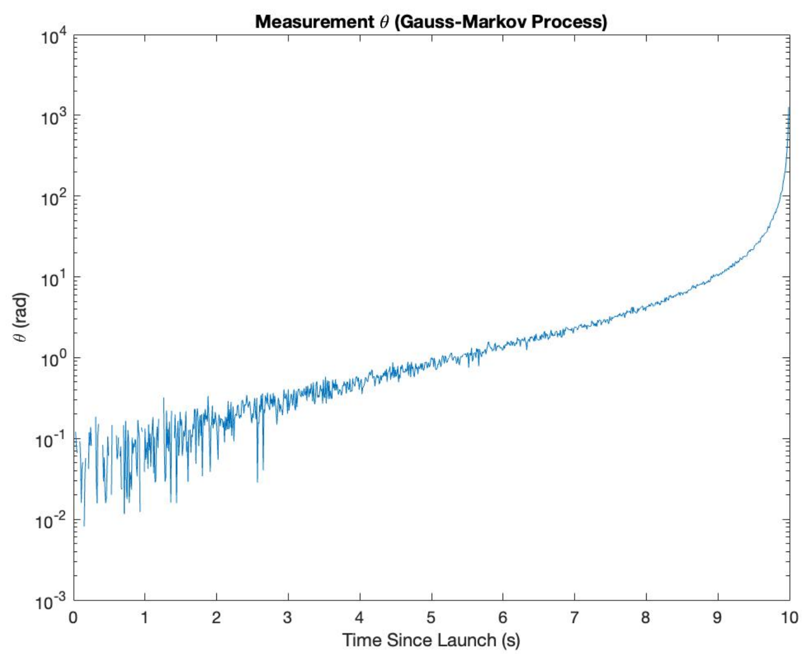

Figure 4: Measurements z for Gauss-Markov process.

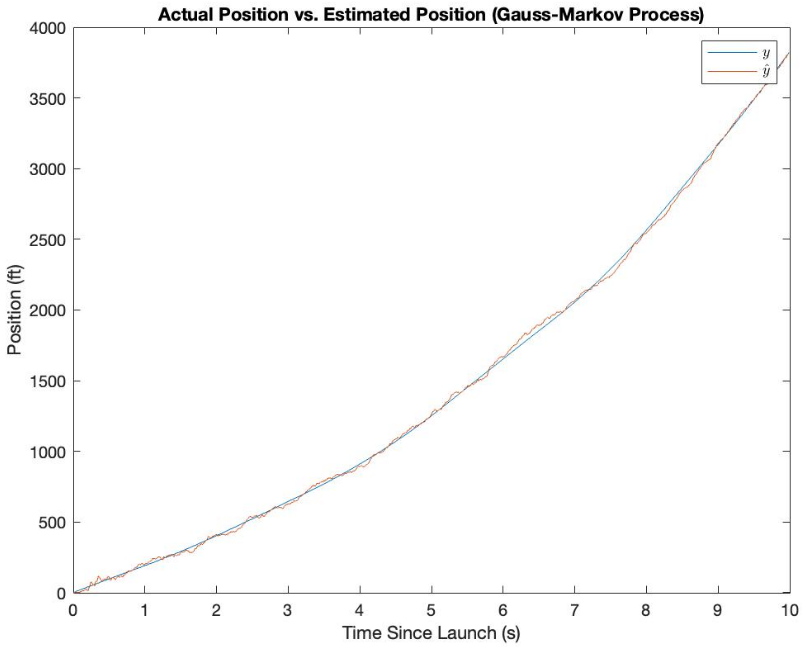

Figure 5: Comparison of actual position and estimated position for Gauss-Markov process.

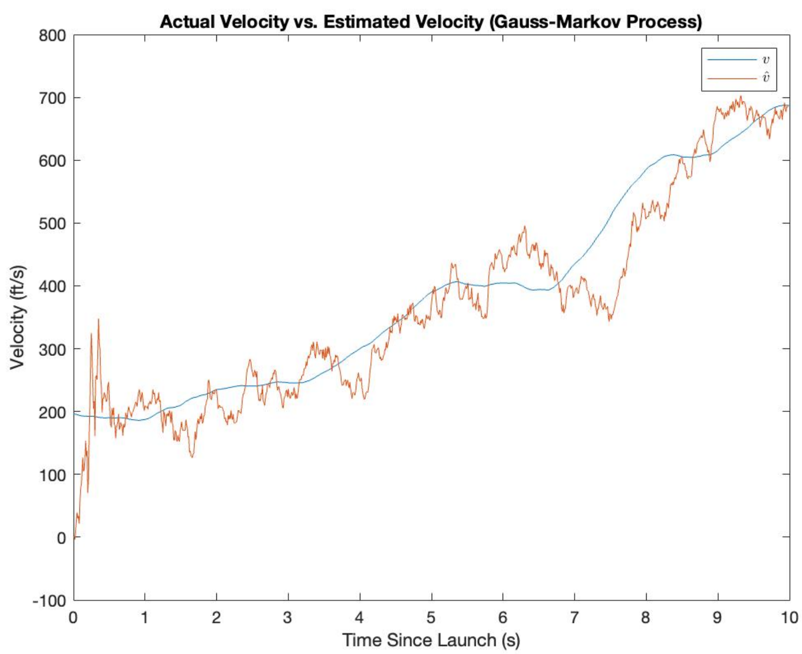

Figure 6: Comparison of actual velocity and estimated velocity for Gauss-Markov process.

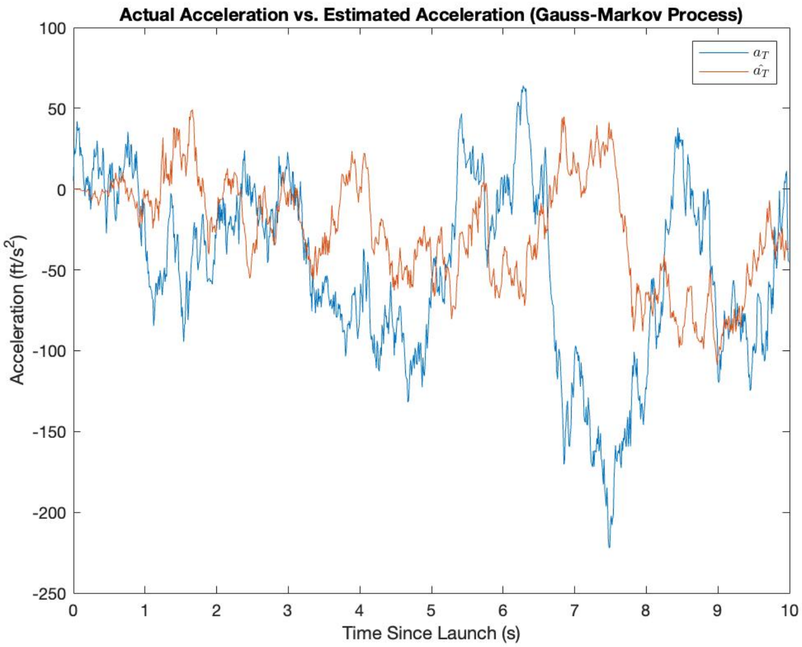

Figure 7: Comparison of actual acceleration and estimated acceleration for Gauss-Markov process.

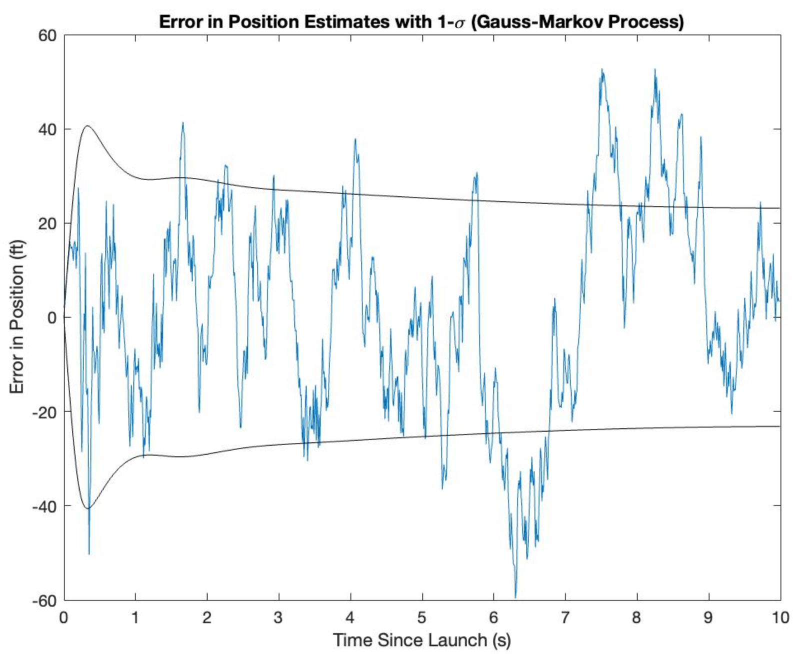

Figure 8: Error in position estimates with 1-σ for Gauss-Markov process.

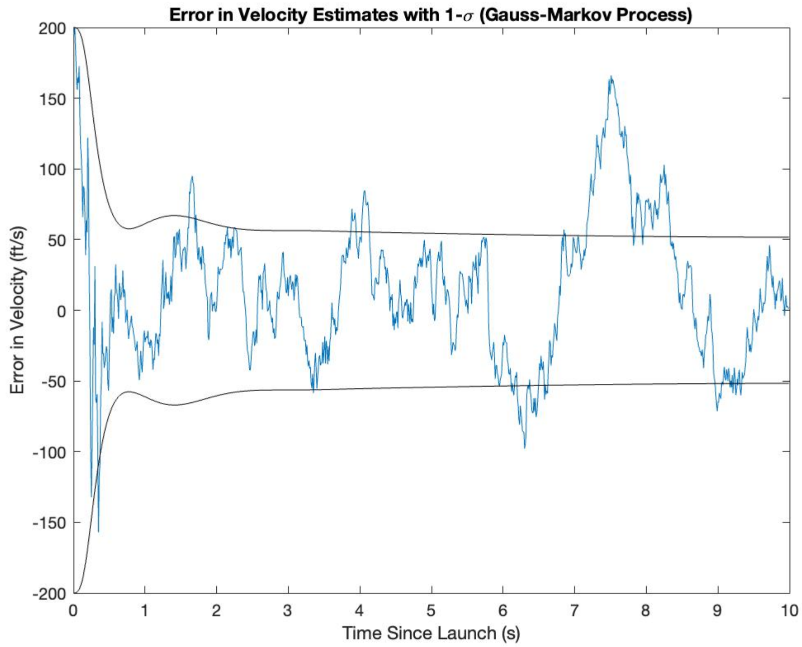

Figure 9: Error in velocity estimates with 1-σ for Gauss-Markov process.

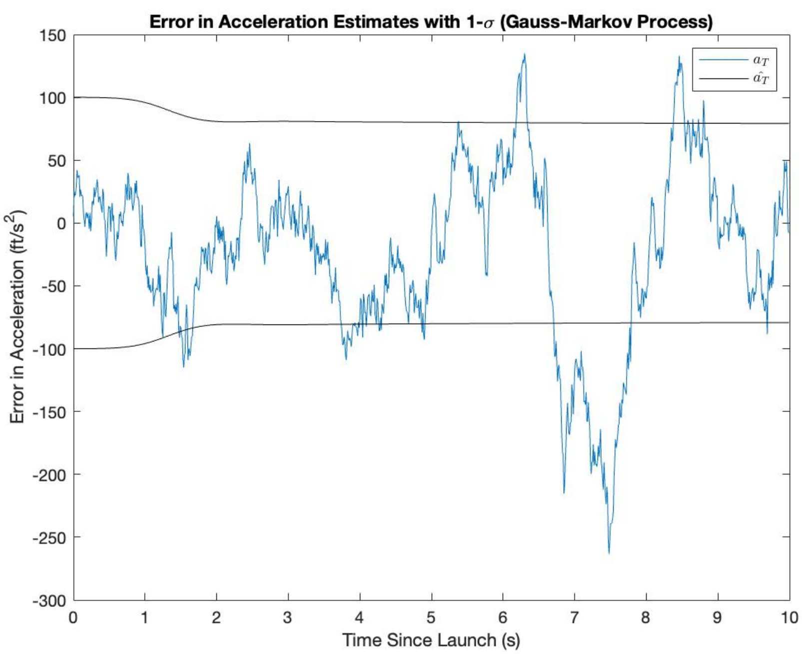

Figure 10: Error in acceleration estimates with 1-σ for Gauss-Markov process.

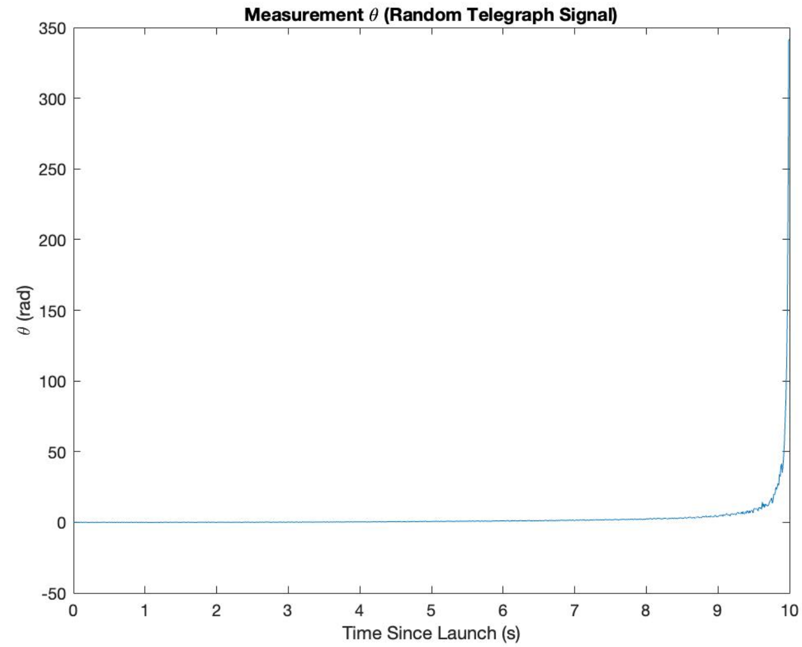

Figure 11: Measurements z for random telegraph signal.

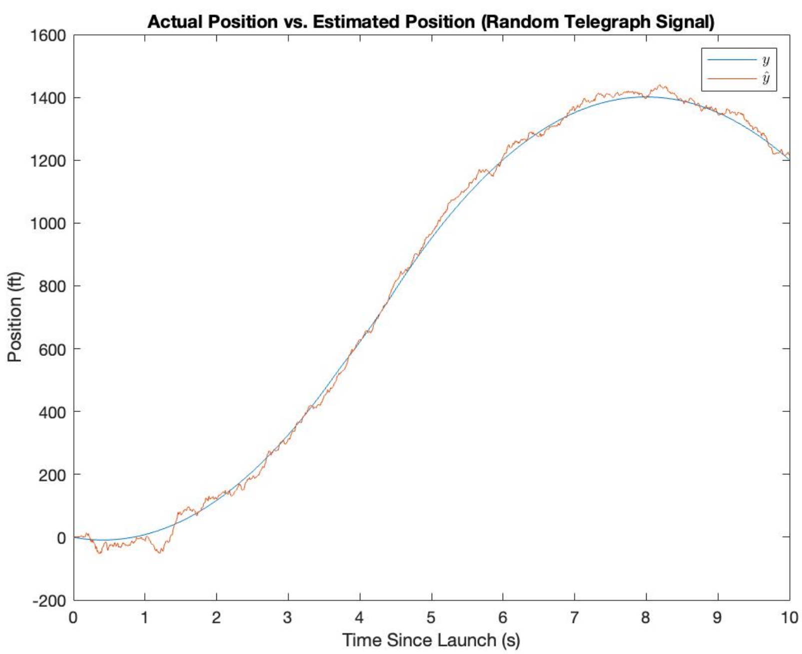

Figure 12: Comparison of actual position and estimated position for random telegraph signal.

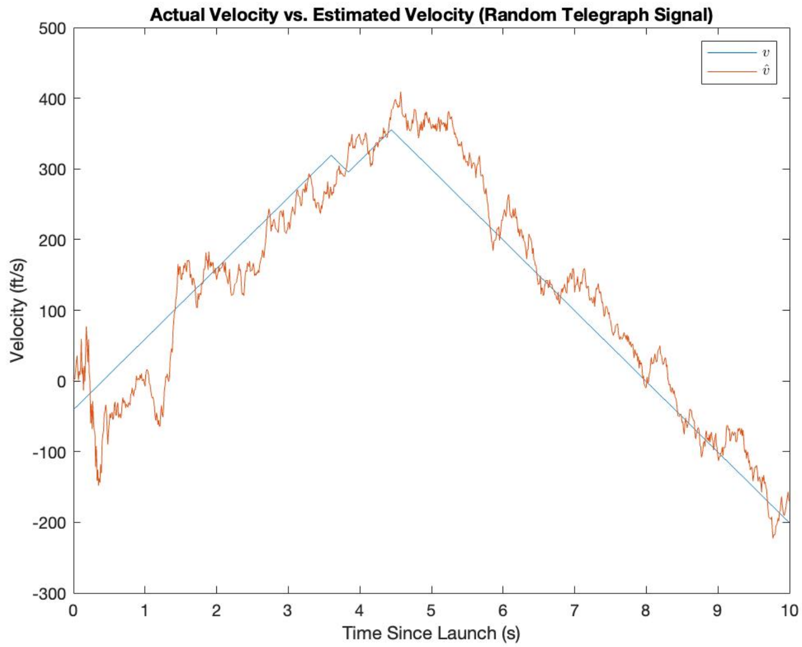

Figure 13: Comparison of actual velocity and estimated velocity for random telegraph signal.

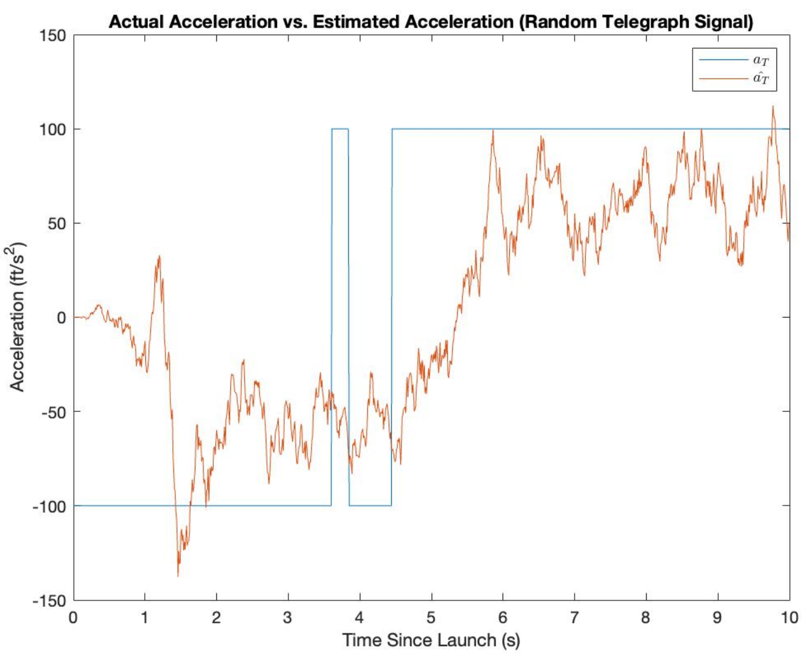

Figure 14: Comparison of actual acceleration and estimated acceleration for random telegraph signal.

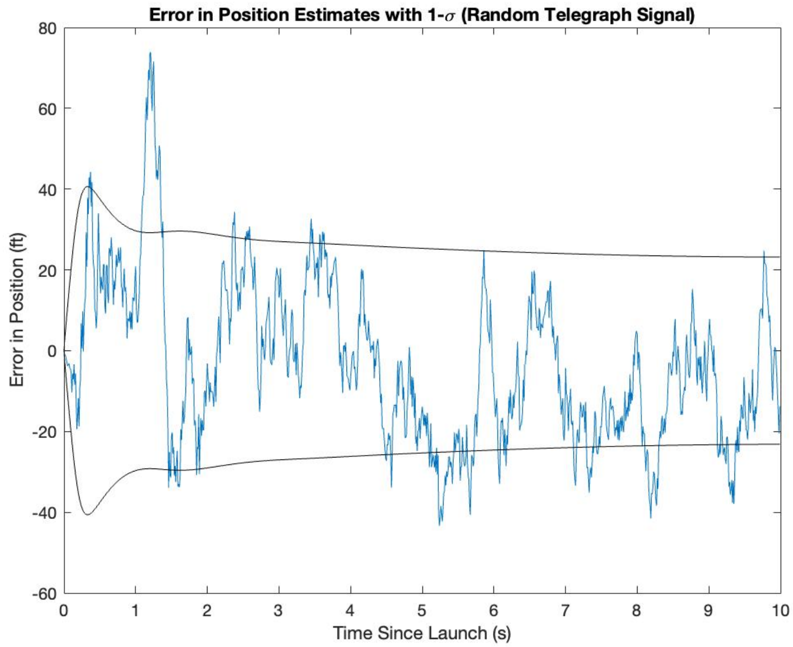

Figure 15: Error in position estimates with 1-σ for random telegraph signal.

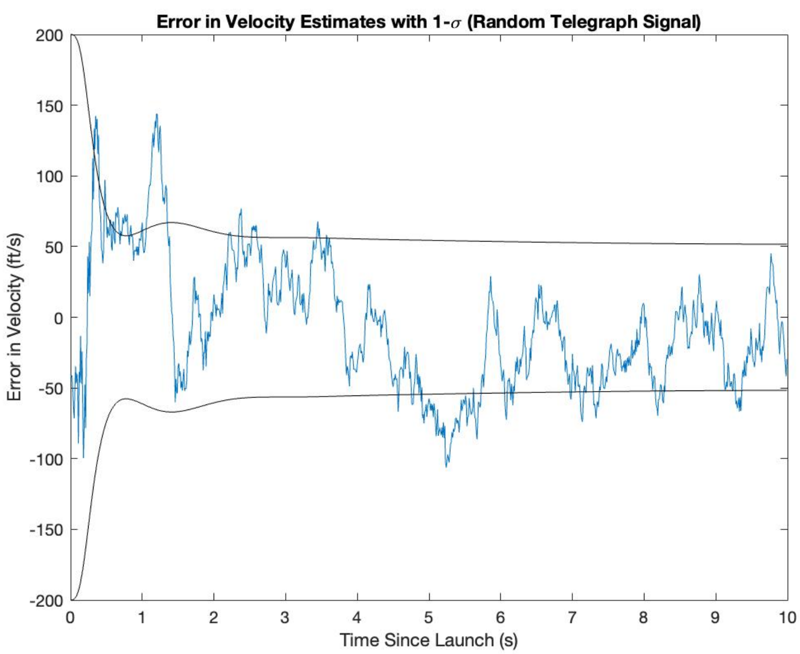

Figure 16: Error in velocity estimates with 1-σ for random telegraph signal.

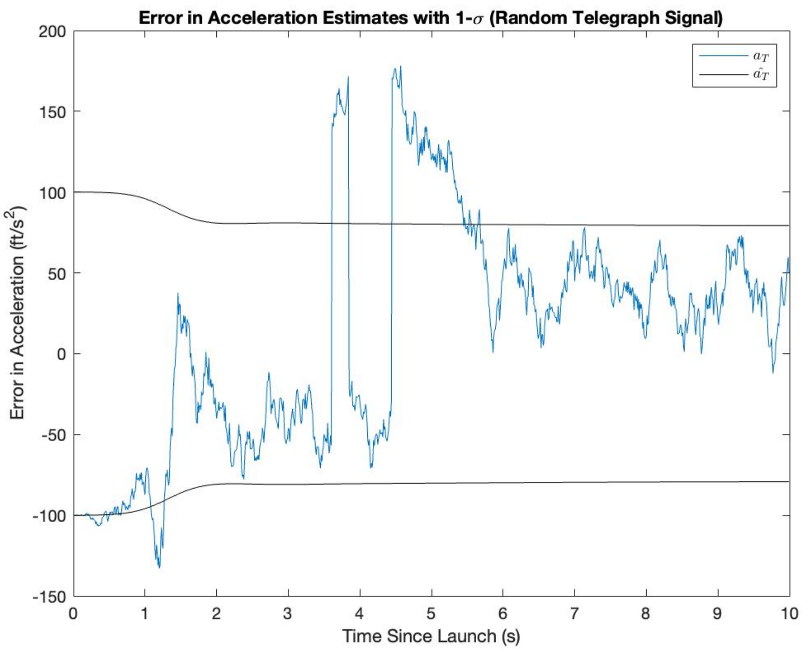

Figure 17: Error in acceleration estimates with 1-σ for random telegraph signal.

5. Monte Carlo Analysis

We perform a Monte Carlo simulation with 1000 realizations of the Kalman filter to understand the actual mean error and error variance through simulation. The error covariance from the simulation is then compared with the error covariance computed by the Kalman filter to understand the performance of the filter.

A Monte Carlo simulation is to be constructed to find the ensemble averages over a set of realizations. Let $e^l(t)$ represent the actual error for realization l, then ensemble average of error $e^l(t)$ which produces the actual mean error is given in equation (8).

$$e^{ave}(t) = \frac{1}{N_{ave}} \sum_{l=1}^{N_{ave}} e^l(t) \tag{8}$$

where $N_{ave} = 1000$ is the number of realizations. It is expected that the $e^{ave}(t) \approx 0$ for all t for an unbiased estimator. The ensemble average producing the actual error variance $P^{ave}(t)$ is given by equation (9).

$$P^{ave}(t) = \frac{1}{N_{ave} - 1} \sum_{l=1}^{N_{ave}} [e^l(t) - e^{ave}(t)][e^l(t) - e^{ave}(t)]^T \tag{9}$$

The matrix $P^{ave}(t)$ should be close to $P(t)$ computed in the Kalman filter algorithm, $P^{ave}(t) - P(t) \approx 0$ for all t to ensure that the Kalman filter model and algorithm are correct. The Kalman filter error variance $P(t)$ and the actual error variance $P^{ave}(t)$ for the Monte Carlo runs corresponding to the Gauss-Markov process is presented in Fig. 18. A similar comparison of $P(t)$ and $P^{ave}(t)$ for random telegraph signal is present in Fig. 19.

Finally, it is shown that the residual process is an white process and the independence of the residuals is shown as in equation (10), where the ensemble average for the correlation of the residuals at time t and t_i is zero when $t \neq t_i$.

$$\frac{1}{N_{ave}} \sum_{l=1}^{N_{ave}} r^l(t_i) r^l(t)^T \approx 0, \forall t \neq t_i \tag{10}$$

To illustrated that the residual process is uncorrelated in time, correlation of the residual process for 7$^{\text{th}}$ second with time t is shown in Fig. 20. In addition the power spectral density (PSD) of the residual process shown in Fig. 21 indicates that it is a white process.

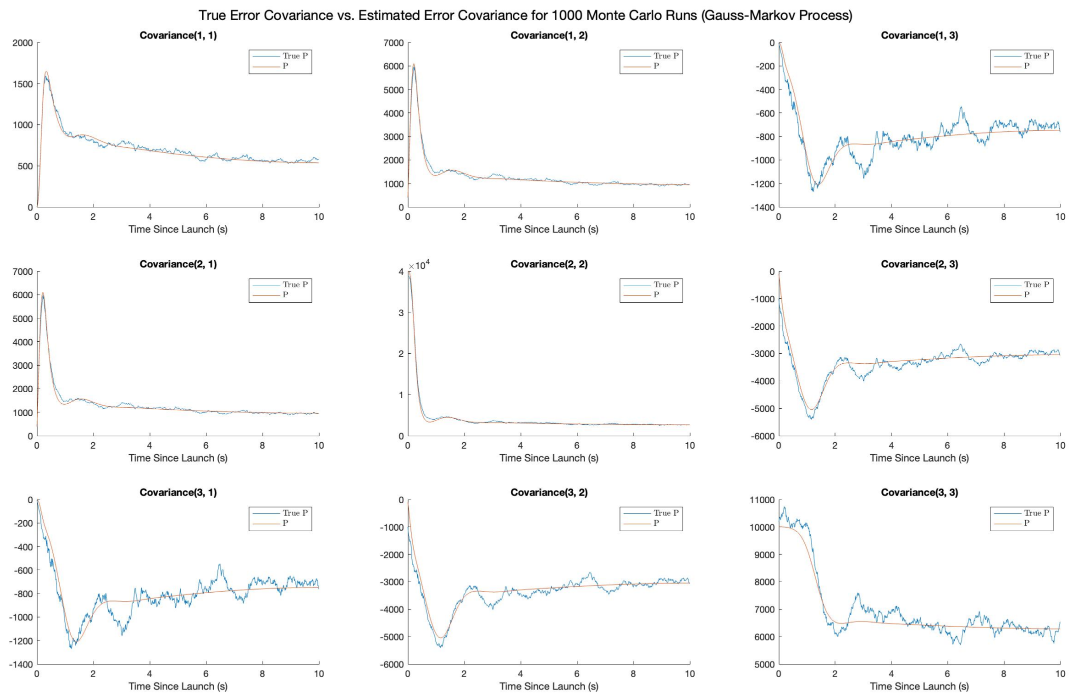

Figure 18: Comparison of actual error covariance $P^{ave}(t)$ with the Kalman filter error covariance estimates $P(t)$ for Gauss-Markov process.

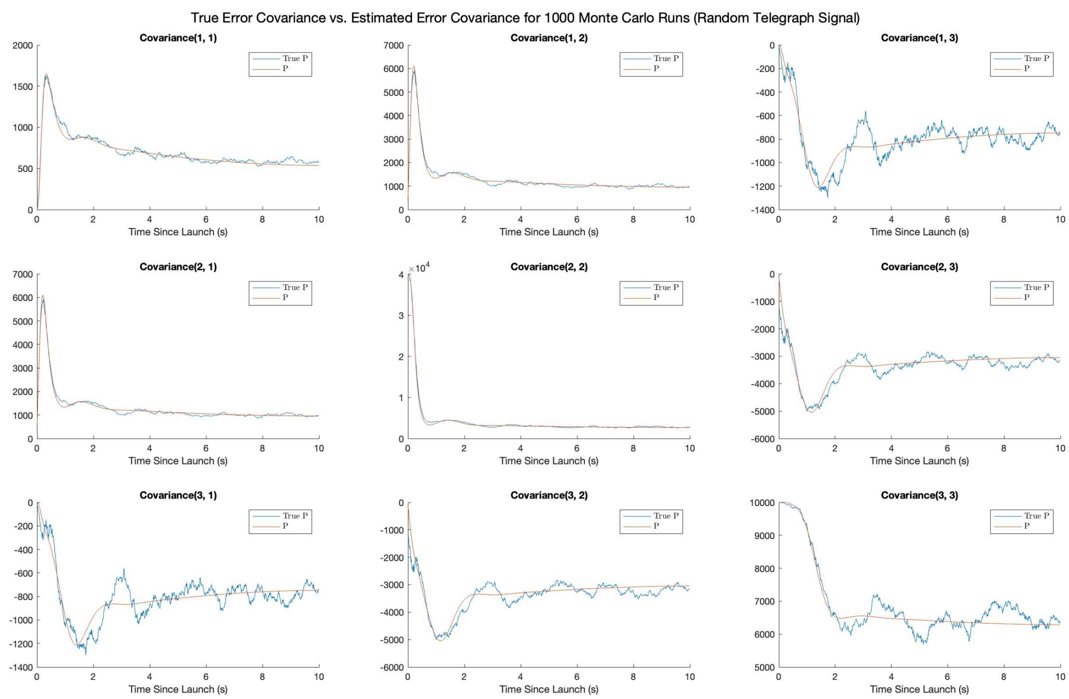

Figure 19: Comparison of actual error covariance $P^{ave}(t)$ with the Kalman filter error covariance estimates $P(t)$ for random telegraph signal.

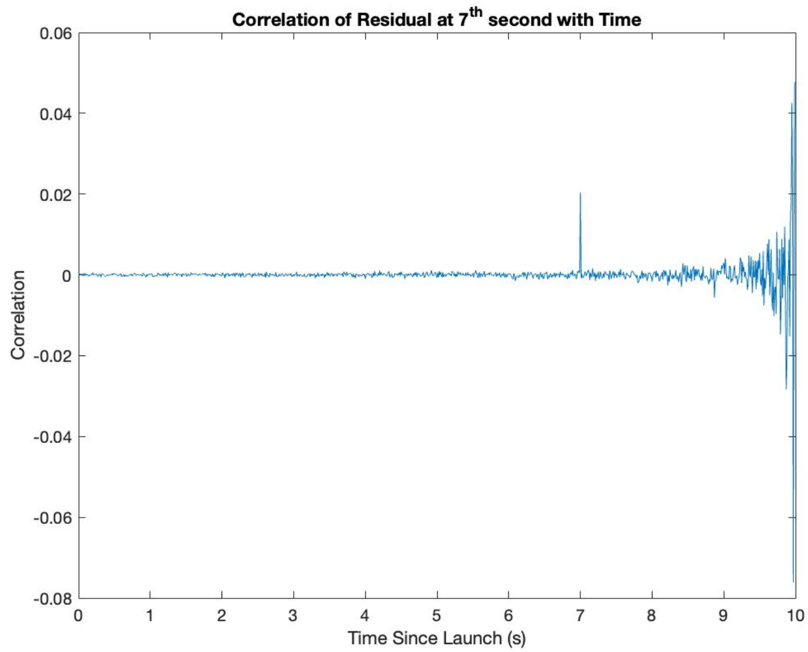

Figure 20: Correlation of the residual process at time $t = 7\,s$ with time.

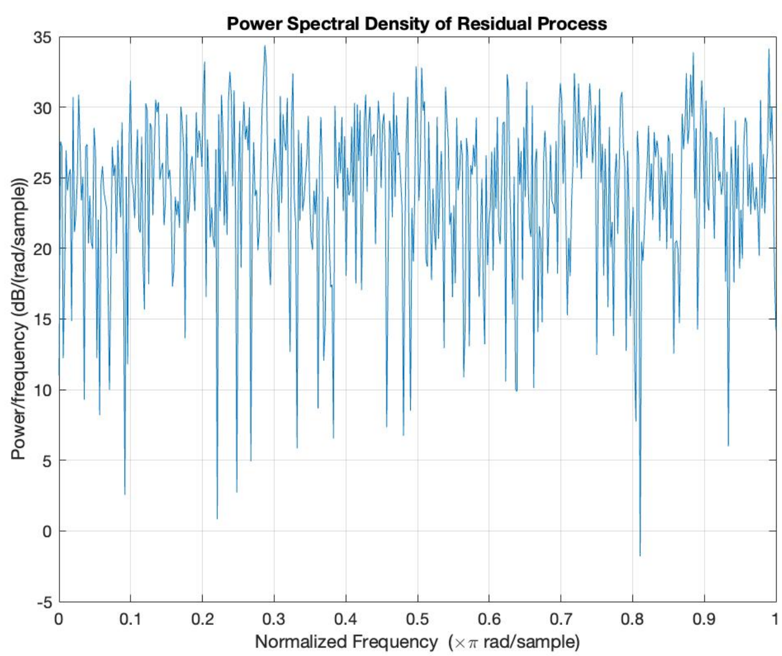

Figure 21: Power spectral density of the residual process shows that it is a white process.

6. Conclusion

In this project Kalman-Bucy filter was used to estimate the lateral position, velocity and acceleration of a missile launched from an aircraft. The estimator was developed on a Gauss-Markov process and tested on an random telegraph signal input with zero mean and similar correlation. This kind of random telegraph signal is more realistic, the simulations show that the estimator performs equally well on this kind of signal. Monte Carlo analysis was performed to check the correctness of the estimator and it can be seen that the estimator behaves well on the random telegraph signal. From the experiments it can be seen that using the theory for Gauss-Markov process, estimators can be designed for a more realistic model of the increment process a_T given by random telegraph signal.

7. MATLAB Code

```
clc;
close all;
clear;

tau = 2;
Y_0_MEAN = 0;
V_0_MEAN = 0;
At_0_MEAN = 0;

V_VAR = 200^2;
At_VAR = 100^2;

Vc = 300;
Tf = 10;
R1 = 15e-6;
R2 = 1.67e-3;

AtTlg = 100;

p_0 = zeros(3, 3);
p_0(2, 2) = V_VAR;
p_0(3, 3) = At_VAR;

dt = 0.01;

ts = linspace(0, Tf, Tf / dt + 1);
ts = ts(1:end-1);

F = [0, 1, 0; 0, 0, -1; 0, 0, -1/tau];
G = [0; 0; 1];
W = At_VAR;

```

```
[filterKs, filterPs] = computePGM(ts, p_0, F, G, W);

n = 1000;
trueXHistory= zeros(n, 3, size(ts, 2));
trueYHistory= zeros(n, 1, size(ts, 2));
xHatHistory= zeros(n, 3, size(ts, 2));
residualHistory = zeros(n, 1, size(ts, 2));

trueXHistoryRTS = zeros(n, 2, size(ts, 2));
trueYHistoryRTS = zeros(n, 1, size(ts, 2));
xHatHistoryRTS = zeros(n, 3, size(ts, 2));
residualHistoryRTS = zeros(n, 1, size(ts, 2));
trueAtHistoryRTS = zeros(n, 1, size(ts, 2));

for i = 1:n
    [trueXs, trueYs] = simDynamics(ts, F, G, W);
    trueXHistory(i, :, :) = trueXs;
    trueYHistory(i, :, :) = trueYs;
    [xHats, residuals] = myKfInnovate(ts, filterKs, trueYs, F);
    xHatHistory(i, :, :) = xHats;
    residualHistory(i, :, :) = residuals;

    [trueXsRTS, trueYsRTS, atRTS] = simTelegraphDynamics(dt, ts, AtTlg, Tf);
    trueXHistoryRTS(i, :, :) = trueXsRTS;
    trueYHistoryRTS(i, :, :) = trueYsRTS;
    trueAtHistoryRTS(i, :, :) = atRTS;
    [xHatsRTS, residualsRTS] = myKfInnovate(ts, filterKs, trueYsRTS, F);
    xHatHistoryRTS(i, :, :) = xHatsRTS;
    residualHistoryRTS(i, :, :) = residualsRTS;
end

[trueErrorVariance] = monteCarloAnalysis(trueXHistory, xHatHistory);
[trueErrorVarianceRTS] = monteCarloAnalysis(cat(2, trueXHistoryRTS, trueAtHistoryRTS) , xHatHistoryRTS);

plotResults1(ts, Tf, filterKs, filterPs, trueErrorVariance)
plotResults2(ts, trueYs, trueXs, xHats, filterPs)
plotResults3(ts, trueYsRTS, cat(1, trueXsRTS, atRTS), xHatsRTS, filterPs)
plotResults4(ts, trueErrorVariance, trueErrorVarianceRTS, filterPs)
plotResults5(ts, dt, residualHistory)
```

```
function [ks, ps] = computePGM(ts, p_0, F, G, W)
    dt = ts(2) - ts(1);
    ps = zeros(1, 3, 3);
    ps(1, :, :) = p_0;

    ks = zeros(3, 1);
    ks(:, 1) = [0; 0; 0];

    R1 = 15e-6;
    R2 = 1.67e-3;
    Vc = 300;
    Tf = 10;

    for t = ts
        V = R1 + R2/((Tf-t)^2);

        H = [1/(Vc*(Tf - t)), 0, 0];
        ks(:, end+1) = squeeze(ps(end, : , :)) * H' / V;

        p_dot = F * squeeze(ps(end, :, :)) + squeeze(ps(end, : , :)) * F' - ks(:, end) * H * squeeze(ps(end, :, :)) + G * W * G';
        ps(end+1, :, :) = squeeze(ps(end, : ,:)) + p_dot * dt;
    end

    ks = ks(:, 2:end);
    ps = ps(2:end, :, :);

end

function [xs, ys] = simDynamics(ts, F, G, W)
    dt = ts(2) - ts(1);

    R1 = 15e-6;
    R2 = 1.67e-3;
    Vc = 300;
    Tf = 10;

    Y_0_MEAN = 0;

    V_0_MEAN = 0;
    V_VAR = 200^2;

    At_0_MEAN = 0;
```

```
    At_VAR = 100^2;

    xs = [[Y_0_MEAN; normrnd(V_0_MEAN, sqrt(V_VAR)); normrnd(
        At_0_MEAN, sqrt(At_VAR))]];
    ys = [0];

    w_at = normrnd(0, sqrt(W/dt), size(ts));
    i = 1;
    V = R1 + R2./((Tf-ts).^2);
    v = normrnd(0, sqrt(V/dt));
    for t = ts
        H = [1/(Vc*(Tf - t)), 0, 0];
        x_dot = F * xs(:, end) + G * w_at(i);
        xs(:, end+1) = xs(:, end) + x_dot*dt;
        ys(end+1) = H * xs(:, end) + v(i);
        i = i + 1;
    end

    xs = xs(:, 2:end);
    ys = ys(2:end);
end

function [xHats, residuals] = myKfInnovate(ts, ks, zs, F)
    dt = ts(2) - ts(1);
    Vc = 300;
    Tf = 10;

    xHats = [[0; 0; 0]];
    residuals = [0];
    i = 1;
    for t = ts
        H = [1/(Vc*(Tf - t)), 0, 0];
        residual = zs(i) - H * xHats(:, end);
        x_dot = F*xHats(:, end) + ks(:, i) * residual;
        xHats(:, end+1) = xHats(:, end) + x_dot * dt;
        residuals(end+1) = residual;
        i = i + 1;
    end

    xHats = xHats(:, 2:end);
    residuals = residuals(2:end);
end

```

```
function trueErrorVariance = monteCarloAnalysis(trueXHistory, xHatHistory)
    error = trueXHistory - xHatHistory;
    meanError = squeeze(mean(error, 1));
    trueErrorVariance = zeros(size(xHatHistory, 3), 3, 3);
    for i = 1:size(xHatHistory, 3)
        for j = 1:size(xHatHistory, 1)
            trueErrorVariance(i, :, :) = squeeze(trueErrorVariance(i, :, :)) + ((error(j, :, i)' - meanError(:, i)) * ((error(j, :, i)' - meanError(:, i))'));
        end
    end
    trueErrorVariance = trueErrorVariance / (size(xHatHistory, 1)-1);

end

function aRTS = generateRTS(dt, Tf, AtTlg)
    t = 0;
    ts = [];
    lambda = 0.25;
    while t < Tf
        t = t - log(unifrnd(0, 1))/lambda;
        ts(end+1) = t;
    end

    ts = unique(roundn(ts, log10(dt)));

    sampleT = linspace(0, Tf, Tf / dt + 1);

    [locA, locB] = ismember(ts, sampleT);
    locB = locB(locB ~= 0);

    switchT = zeros(size(sampleT));
    switchT(locB) = 1;

    aRTS = zeros(size(sampleT)-1);
    flip = binornd(1, 0.5);
    if(flip)
        sign = 1;
    else
        sign = -1;
    end

    for i = 1:size(aRTS, 2)
```

```
        if(switchT(i) == 1)
            sign = sign * -1;
        end
        aRTS(i) = sign * AtTlg;
    end
end

function [xs, ys, atRTS] = simTelegraphDynamics(dt, ts, AtTlg, Tf)
    atRTS = generateRTS(dt, Tf, AtTlg);
    F = [0, 1; 0, 0];
    G = [0; -1];

    dt = ts(2) - ts(1);

    R1 = 15e-6;
    R2 = 1.67e-3;
    Vc = 300;
    Tf = 10;

    Y_0_MEAN = 0;

    V_0_MEAN = 0;
    V_VAR = 200^2;

    xs = [[Y_0_MEAN; normrnd(V_0_MEAN, sqrt(V_VAR))]];
    ys = [0];

    i = 1;
    V = R1 + R2./((Tf-ts).^2);
    v = normrnd(0, sqrt(V/dt));
    for t = ts
        H = [1/(Vc*(Tf - t)), 0];
        x_dot = F * xs(:, end) + G * atRTS(i);
        xs(:, end+1) = xs(:, end) + x_dot*dt;
        ys(end+1) = H * xs(:, end) + v(i);
        i = i + 1;
    end

    xs = xs(:, 2:end);
    ys = ys(2:end);
end

function residualCorrelations = residualAnalysis(timeToAnalysis,
    residualHistory, dt)
```

```
    residualT = residualHistory(:, :, (timeToAnalysis/dt)+1);
    residualCorrelations = zeros(size(residualHistory, 2), size(
       residualHistory, 3));
    for i = 1:size(residualHistory, 1)
        for j = 1:size(residualHistory, 3)
            residualCorrelations(j) = residualCorrelations(j) + (
               residualHistory(i, 1, j) * residualT(i, 1)');
        end
    end
    residualCorrelations = residualCorrelations / size(
       residualHistory, 1);
end

function plotResults1(ts, Tf, filterKs, filterPs, trueErrorVariance)
    figure(1)
    ax1 = axes;
    plot(Tf - ts, filterKs(1, :))
    hold on
    plot(Tf - ts, filterKs(2, :))
    plot(Tf - ts, filterKs(3, :))
    hold off
    set(ax1, 'Xdir', 'reverse')
    xlabel('Time-to-Go (s)')
    ylabel('Kalman Filter Gains')
    legend('K_{1}', 'K_{2}', 'K_{3}')
    set(get(gca, 'Title'), 'String', 'Kalman Filter Gains');

    figure(2)
    ax1 = axes;
    plot(Tf - ts, sqrt(filterPs(:, 1, 1)))
    hold on
    plot(Tf - ts, sqrt(filterPs(:, 2, 2)))
    plot(Tf - ts, sqrt(filterPs(:, 3, 3)))
    plot(Tf - ts, sqrt(trueErrorVariance(:, 1, 1)))
    plot(Tf - ts, sqrt(trueErrorVariance(:, 2, 2)))
    plot(Tf - ts, sqrt(trueErrorVariance(:, 3, 3)))
    hold off
    set(ax1, 'Xdir', 'reverse')
    xlabel('Time-to-Go (s)')
    ylabel('Standard Deviation of State Error')
    legend('\sigma_{y}', '\sigma_{v}', '\sigma_{a_{T}}', 'Actual \
       sigma_{y}', 'Actual \sigma_{v}', 'Actual \sigma_{a_{T}}')
    set(get(gca, 'Title'), 'String', 'Kalman Filter Estimated State
       RMS Error P(t)^{1/2} vs. Actual State RMS Error');
```

```
    figure(21)
    ax1 = axes;
    plot(Tf - ts, filterPs(:, 1, 1))
    hold on
    plot(Tf - ts, filterPs(:, 2, 2))
    plot(Tf - ts, filterPs(:, 3, 3))
    hold off
    set(ax1, 'Xdir', 'reverse')
    xlabel('Time-to-Go (s)')
    ylabel('Estimates of State Error Variance')
    legend('\sigma_{y}^{2}', '\sigma_{v}^{2}', '\sigma_{a_{T}}^{2}')
    set(get(gca, 'Title'), 'String', 'Kalman Filter State Error Variance P(t)');

end

function plotResults2(ts, trueYs, trueXs, xHats, filterPs)
    figure(3)
    plot(ts, trueYs)
    xlabel('Time Since Launch (s)')
    ylabel('\theta (rad)')
    set(gca, 'YScale', 'log')
    set(get(gca, 'Title'), 'String', 'Measurement \theta (Gauss-Markov Process)');

    figure(4)
    plot(ts, trueXs(1, :))
    hold on
    plot(ts, xHats(1, :))
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Position (ft)')
    set(legend('$y$', '$\hat{y}$'),'Interpreter','Latex')
    set(get(gca, 'Title'), 'String', 'Actual Position vs. Estimated Position (Gauss-Markov Process)');

    figure(5)
    plot(ts, trueXs(2, :))
    hold on
    plot(ts, xHats(2, :))
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Velocity (ft/s)')
```

```
    set(legend('$v$', '$\hat{v}$'),'Interpreter','Latex')
    set(get(gca, 'Title'), 'String', 'Actual Velocity vs. Estimated
       Velocity (Gauss-Markov Process)');

    figure(6)
    plot(ts, trueXs(3, :))
    hold on
    plot(ts, xHats(3, :))
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Acceleration (ft/s^2)')
    set(legend('$a_{T}$', '$\hat{a_{T}}$'),'Interpreter','Latex')
    set(get(gca, 'Title'), 'String', 'Actual Acceleration vs.
       Estimated Acceleration (Gauss-Markov Process)');

    figure(7)
    P1 = sqrt(filterPs(:, 1, 1)');
    plot(ts, trueXs(1, :) - xHats(1, :))
    hold on
    plot(ts, P1, 'black')
    plot(ts, - P1, 'black')
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Error in Position (ft)')
    set(get(gca, 'Title'), 'String', 'Error in Position Estimates
       with 1-\sigma (Gauss-Markov Process)');

    figure(8)
    P2 = sqrt(filterPs(:, 2, 2)');
    plot(ts, trueXs(2, :) - xHats(2, :))
    hold on
    plot(ts, P2, 'black')
    plot(ts, - P2, 'black')
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Error in Velocity (ft/s)')
    set(get(gca, 'Title'), 'String', 'Error in Velocity Estimates
       with 1-\sigma (Gauss-Markov Process)');

    figure(9)
    P3 = sqrt(filterPs(:, 3, 3)');
    plot(ts, trueXs(3, :) - xHats(3, :))
    hold on
    plot(ts, P3, 'black')
```

```
    plot(ts, - P3, 'black')
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Error in Acceleration (ft/s^2)')
    set(legend('$a_{T}$', '$\hat{a_{T}}$'),'Interpreter','Latex')
    set(get(gca, 'Title'), 'String', 'Error in Acceleration Estimates
        with 1-\sigma (Gauss-Markov Process)');

end

function plotResults3(ts, trueYs, trueXs, xHats, filterPs)
    figure(10)
    plot(ts, trueYs)
    xlabel('Time Since Launch (s)')
    ylabel('\theta (rad)')
   % set(gca, 'YScale', 'log')
    set(get(gca, 'Title'), 'String', 'Measurement \theta (Random
       Telegraph Signal)');

    figure(11)
    plot(ts, trueXs(1, :))
    hold on
    plot(ts, xHats(1, :))
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Position (ft)')
    set(legend('$y$', '$\hat{y}$'),'Interpreter','Latex')
    set(get(gca, 'Title'), 'String', 'Actual Position vs. Estimated
       Position (Random Telegraph Signal)');

    figure(12)
    plot(ts, trueXs(2, :))
    hold on
    plot(ts, xHats(2, :))
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Velocity (ft/s)')
    set(legend('$v$', '$\hat{v}$'),'Interpreter','Latex')
    set(get(gca, 'Title'), 'String', 'Actual Velocity vs. Estimated
       Velocity (Random Telegraph Signal)');

    figure(13)
    plot(ts, trueXs(3, :))
    hold on
```

```
    plot(ts, xHats(3, :))
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Acceleration (ft/s^2)')
    set(legend('$a_{T}$', '$\hat{a_{T}}$'),'Interpreter','Latex')
    set(get(gca, 'Title'), 'String', 'Actual Acceleration vs. Estimated Acceleration (Random Telegraph Signal)');

    figure(14)
    P1 = sqrt(filterPs(:, 1, 1)');
    plot(ts, trueXs(1, :) - xHats(1, :))
    hold on
    plot(ts, P1, 'black')
    plot(ts, -P1, 'black')
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Error in Position (ft)')
    set(get(gca, 'Title'), 'String', 'Error in Position Estimates with 1-\sigma (Random Telegraph Signal)');

    figure(15)
    P2 = sqrt(filterPs(:, 2, 2)');
    plot(ts, trueXs(2, :) - xHats(2, :))
    hold on
    plot(ts, P2, 'black')
    plot(ts, -P2, 'black')
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Error in Velocity (ft/s)')
    set(get(gca, 'Title'), 'String', 'Error in Velocity Estimates with 1-\sigma (Random Telegraph Signal)');

    figure(16)
    P3 = sqrt(filterPs(:, 3, 3)');
    plot(ts, trueXs(3, :) - xHats(3, :))
    hold on
    plot(ts, P3, 'black')
    plot(ts, -P3, 'black')
    hold off
    xlabel('Time Since Launch (s)')
    ylabel('Error in Acceleration (ft/s^2)')
    set(legend('$a_{T}$', '$\hat{a_{T}}$'),'Interpreter','Latex')
    set(get(gca, 'Title'), 'String', 'Error in Acceleration Estimates with 1-\sigma (Random Telegraph Signal)');
```

```
end

function plotResults4(ts, trueErrorVariance, trueErrorVarianceRTS,
    filterPs)
    figure(17)
    subplot(3, 3, 1)
    hold on
    plot(ts, trueErrorVariance(:, 1, 1))
    plot(ts, filterPs(:, 1, 1))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(1, 1)')

    subplot(3, 3, 2)
    hold on
    plot(ts, trueErrorVariance(:, 1, 2))
    plot(ts, filterPs(:, 1, 2))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(1, 2)')

    subplot(3, 3, 3)
    hold on
    plot(ts, trueErrorVariance(:, 1, 3))
    plot(ts, filterPs(:, 1, 3))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(1, 3)')

    subplot(3, 3, 4)
    hold on
    plot(ts, trueErrorVariance(:, 2, 1))
    plot(ts, filterPs(:, 2, 1))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(2, 1)')

    subplot(3, 3, 5)
    hold on
```

```
    plot(ts, trueErrorVariance(:, 2, 2))
    plot(ts, filterPs(:, 2, 2))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(2, 2)')

    subplot(3, 3, 6)
    hold on
    plot(ts, trueErrorVariance(:, 2, 3))
    plot(ts, filterPs(:, 2, 3))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(2, 3)')

    subplot(3, 3, 7)
    hold on
    plot(ts, trueErrorVariance(:, 3, 1))
    plot(ts, filterPs(:, 3, 1))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(3, 1)')

    subplot(3, 3, 8)
    hold on
    plot(ts, trueErrorVariance(:, 3, 2))
    plot(ts, filterPs(:, 3, 2))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(3, 2)')

    subplot(3, 3, 9)
    hold on
    plot(ts, trueErrorVariance(:, 3, 3))
    plot(ts, filterPs(:, 3, 3))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(3, 3)')
```

```
    sgtitle('True Error Covariance vs. Estimated Error Covariance for 1000 Monte Carlo Runs (Gauss-Markov Process)')

    figure(18)
    subplot(3, 3, 1)
    hold on
    plot(ts, trueErrorVarianceRTS(:, 1, 1))
    plot(ts, filterPs(:, 1, 1))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(1, 1)')

    subplot(3, 3, 2)
    hold on
    plot(ts, trueErrorVarianceRTS(:, 1, 2))
    plot(ts, filterPs(:, 1, 2))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(1, 2)')

    subplot(3, 3, 3)
    hold on
    plot(ts, trueErrorVarianceRTS(:, 1, 3))
    plot(ts, filterPs(:, 1, 3))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(1, 3)')

    subplot(3, 3, 4)
    hold on
    plot(ts, trueErrorVarianceRTS(:, 2, 1))
    plot(ts, filterPs(:, 2, 1))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(2, 1)')

    subplot(3, 3, 5)
    hold on
    plot(ts, trueErrorVarianceRTS(:, 2, 2))
    plot(ts, filterPs(:, 2, 2))
```

```
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(2, 2)')

    subplot(3, 3, 6)
    hold on
    plot(ts, trueErrorVarianceRTS(:, 2, 3))
    plot(ts, filterPs(:, 2, 3))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(2, 3)')

    subplot(3, 3, 7)
    hold on
    plot(ts, trueErrorVarianceRTS(:, 3, 1))
    plot(ts, filterPs(:, 3, 1))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(3, 1)')

    subplot(3, 3, 8)
    hold on
    plot(ts, trueErrorVarianceRTS(:, 3, 2))
    plot(ts, filterPs(:, 3, 2))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(3, 2)')

    subplot(3, 3, 9)
    hold on
    plot(ts, trueErrorVarianceRTS(:, 3, 3))
    plot(ts, filterPs(:, 3, 3))
    hold off
    xlabel('Time Since Launch (s)')
    set(legend('True P', 'P'),'Interpreter','Latex')
    title('Covariance(3, 3)')

    sgtitle('True Error Covariance vs. Estimated Error Covariance for
        1000 Monte Carlo Runs (Random Telegraph Signal)')
end
```

```
function plotResults5(ts, dt, residualHistory)
    residualCorrelations = residualAnalysis(7, residualHistory, dt);

    figure(19)
    plot(ts, residualCorrelations)
    xlabel('Time Since Launch (s)')
    ylabel('Correlation')
    title('Correlation of Residual at 7^{th} second with Time')
    Tf = 10;
    R1 = 15e-6;
    R2 = 1.67e-3;
    V = R1 + R2./((Tf-ts).^2);

    figure(20)
    periodogram(squeeze(mean(residualHistory, 1))./V')
    title('Power Spectral Density of Residual Process')

end
```