Calibration of an Accelerometer Using GPS Measurements

James Tseng

MAE C175A / 271A

10 December 2021

1 Introduction

1.1 Problem Definition

There is a vehicle moving in one degree of freedom that is oscillating in acceleration. We are given an accelerometer that is able to measure the acceleration of the vehicle, with noise and a bias. We are also given GPS measurements of the vehicle's position and velocity, with noise, at sparser sampling rates than the accelerometer. The goal is to derive and implement an estimator that fuses both the accelerometer and GPS information to estimate the vehicle position, velocity, and also the accelerometer bias.

1.2 Prompted Solution

The estimator to be implemented is a discrete Kalman filter minimum variance estimator. First, a system dynamics model is derived. Since the accelerometer model is sufficiently fast, we use the sample rate of the accelerometer as the simulation rate. This allows us to combine the dynamics of the model with the accelerometer measurement and noise. We then derive the measurement equation for the GPS data. Then, we form the four equations for the discrete Kalman filter to be used in the simulation.

To ensure that the implemented filter works as intended across a range of possible inputs, a Monte-Carlo simulation is conducted on the filter using various initial conditions. We validate the performance of the filter by inspecting the position, velocity, bias estimates, and respective errors, variances across the ensemble of the Monte-Carlo simulation, and the upholding of the theoretical orthogonality properties.

2 Theory and Algorithm

Here we derive the dynamics model, measurement equations, Kalman filter equations, and the equations for filter validation.

2.1 Truth Model

We are given that the acceleration equation, and by integration, we obtain

$$\begin{aligned}
a(t) &= a\sin(\omega t) \\
v(t) &= v(0) + \frac{a}{\omega} - \frac{a}{\omega}\cos(\omega t) \\
p(t) &= p(0) + (v(0) + \frac{a}{\omega})t - \frac{a}{\omega^2}\sin(\omega t)
\end{aligned}$$

where $v(0)$ and $p(0)$ are randomly chosen across the Monte-Carlo simulation, characterized by $v(0) \sim N(100 \text{ m/s}, 1 \text{ (m/s)}^2)$ and $p(0) \sim N(0 \text{ m}, 100 \text{ m}^2)$.

2.2 Accelerometer Model

Since our accelerometer measurements are discrete and with noise, we can model this using our truth model but add in noise and the accelerometer bias and "integrate" via the Euler method for each time step t_i

$$\begin{aligned}
a_c(t_i) &= a(t_i) + b + \omega(t_i) \\
v_c(t_{i+1}) &= v_c((t_i) + a_c(t_i)\Delta t \\
p_c(t_{i+1}) &= p_c(t_i) + v_c(t_i)\Delta t + a_c(t_i)\frac{\Delta t^2}{2}
\end{aligned}$$

where b is the bias, ω is the process noise, and are respectively characterized by $b \sim N(0 \text{ m}, 0.01 \text{ (m/s}^2)^2)$ and $\omega \sim N(0 \text{ m}, 0.0004 \text{ (m/s}^2)^2)$. We choose the initial conditions $v_c(0) = \bar{v}(0) = 100$ m/s and $p_c(0) = \bar{p}(0) = 0$ m.

2.3 Dynamics Model

We utilize the accelerometer at each time step of our simulation calculation, therefore, we combine the dynamics model with the accelerometer. We first model our estimate equations using the truth model as

$$\begin{aligned}
v_E(t_{i+1}) &= v_E(t_i) + a(t_i)\Delta t \\
p_E(t_{i+1}) &= p_E(t_i) + v_E(t_i)\Delta t + a(t_i)\frac{\Delta t^2}{2}
\end{aligned}$$

In order to obtain our dynamics model that is independent from the acceleration profile, we subtract the accelerometer model from this estimate to obtain $\partial v_E(t_i)$ and $\partial p_E(t_i)$.

$$\begin{aligned}
\partial v_E(t_{i+1}) &= v_E(t_{i+1}) - v_c(t_{i+1}) \\
&= v_E(t_i) + a(t_i)\Delta t - v_c((t_i) - (a(t_i) + b + \omega(t_i))\,\Delta t \\
&= \partial v_E(t_i) - b\Delta t - \omega(t_i)\Delta t \\
\partial p_E(t_{i+1}) &= p_E(t_{i+1}) - p_c(t_{i+1}) \\
&= p_E(t_i) + v_E(t_i)\Delta t + a(t_i)\frac{\Delta t^2}{2} - p_c(t_i) - v_c(t_i)\Delta t - (a(t_i) + b + \omega(t_i))\,\frac{\Delta t^2}{2} \\
&= \partial p_E(t_i) + \partial v_E(t_i)\Delta t - b\frac{\Delta t^2}{2} - \frac{\Delta t^2}{2}\omega(t_i)
\end{aligned}$$

Organize the equations to use in our state-space stochastic discrete time system, where we track $\partial v_E(t_i)$, $\partial p_E(t_i)$, and b.

$$x(t_{i+1}) = \begin{bmatrix} \partial p_E(t_{i+1}) \\ \partial v_E(t_{i+1}) \\ b(t_{i+1}) \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & -\frac{\Delta t^2}{2} \\ 0 & 1 & -\Delta t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \partial p_E(t_i) \\ \partial v_E(t_i) \\ b(t_i) \end{bmatrix} - \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \\ 0 \end{bmatrix} \omega(t_i) \tag{1}$$

2.4 Measurement Equations

The GPS provides position and velocity measurements, which are modeled by adding noise to the truth model

$$\begin{aligned}
z^p(t_i) &= p(t_i) + \eta^p(t_i) \\
z^v(t_i) &= v(t_i) + \eta^v(t_i)
\end{aligned}$$

where η is the process noise, characterized by $\left[\begin{smallmatrix} \eta^p \\ \eta^v \end{smallmatrix}\right] \sim N(\left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right], \left[\begin{smallmatrix} 1m^2 & 0 \\ 0 & 0.0016(m/s)^2 \end{smallmatrix}\right])$.
We can rewrite the equations in terms of ∂p and ∂v by subtracting p_c and v_c from the above equations to obtain

$$z(t_i) = \begin{bmatrix} \partial z^p(t_i) \\ \partial z^v(t_i) \end{bmatrix} = \begin{bmatrix} \partial p(t_i) \\ \partial v(t_i) \end{bmatrix} + \begin{bmatrix} \eta^p(t_i) \\ \eta^v(t_i) \end{bmatrix}$$

2.5 Discrete Kalman Filter

We first define some notations for constants used in the Kalman filter to simply the presentation of the equations.

- Φ, the constant transition matrix, is derived above in equation (1) as $\Phi = \begin{bmatrix} 1 & \Delta t & -\frac{\Delta t^2}{2} \\ 0 & 1 & -\Delta t \\ 0 & 0 & 1 \end{bmatrix}$
- Γ, the constant noise channel vector, is derived above in equation (1) as $\Gamma = \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \\ 0 \end{bmatrix}$

- H, the constant observation matrix that maps the state vector x size 3×1 to the measurement vector z size 2×1 through $z = Hx + \nu$ is $H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

Our states and covariances are tracked *a priori* and *posteriori* as $\bar{x}(t_i), M(t_i)$ and $\hat{x}(t_i), P(t_i)$, respectively, where M, P are size 3×3 and that M_{ii}, P_{ii} are the variances for each state tracked.

Thus, the Kalman filter equations to propagate the state are

$$\bar{x}(t_{i+1}) = \Phi \hat{x}(t_i) \tag{2}$$

$$M(t_{i+1}) = \Phi P(t_i) \Phi^T + \Gamma W \Gamma^T \tag{3}$$

where W, scalar, is the variance of ω noise. Notice that equation (2) is similar to Equation (1), without the process noise, due to the fact that the noise is zero mean, thus the mean $\bar{x}$ does not include the ω term. And to update the state are

$$K(t_i) = M(t_i) H^T \left(H M(t_i) H^T + V \right)^{-1} \tag{4}$$

$$\hat{x}(t_{i+1}) = \bar{x}(t_i) + K(t_i) \left(z(t_i) - H \bar{x}(t_i) \right) \tag{5}$$

$$P(t_{i+1}) = M(t_i) - K(t_i) H M(t_i) \tag{6}$$

where V, size 2×2, is the variance of η noise.

However, the update equations are not utilized every simulation loop as the GPS signal is received at sparser increments than the accelerometer. Therefore, $\hat{x}$ and P are not updated for every simulation time step, meaning the propagation equations (2) and (3) will output the same value for the duration between GPS measurements, which is not ideal. Our implementation of the Kalman filter aims to alleviate that by continuing to propagate the *a priori* estimate for the entire duration between updates. The new equations to propagate the state are

$$\bar{x}(t_{i+1}) = \Phi \bar{x}(t_i) \tag{7}$$

$$M(t_{i+1}) = \Phi M(t_i) \Phi^T + \Gamma W \Gamma^T \tag{8}$$

As such, the covariance M will grow as propagation continues since the filter becomes less certain of the estimate over time as no new measurements are received.

2.6 Error Validation

We define the estimation error of our state from the truth model for each time step as

$$\bar{e}(t_i) = \partial x(t_i) - \partial \bar{x}(t_i) = \begin{bmatrix} p(t_i) - p_c(t_i) \\ v(t_i) - v_c(t_i) \\ b(t_i) \end{bmatrix} - \begin{bmatrix} \bar{p}(t_i) - p_c(t_i) \\ \bar{v}(t_i) - v_c(t_i) \\ \bar{b}(t_i) \end{bmatrix} = \begin{bmatrix} p(t_i) - \bar{p}(t_i) \\ v(t_i) - \bar{v}(t_i) \\ b(t_i) - \bar{b}(t_i) \end{bmatrix} \tag{9}$$

Now, we focus on the validation as an ensemble of realizations across the Monte-Carlo simulation for a large sample size.

Using this, we can define an ensemble average error for each time steps across all Monte-Carlo simulations as

$$e^{ave}(t_i) = \frac{1}{N_{ave}} \sum_{l=1}^{N_{ave}} e^l(t_i) \tag{10}$$

where N_{ave} is the number of Monte-Carlo simulation runs.
We next define the ensemble average error covariance as

$$P^{ave}(t_i) = \frac{1}{N_{ave} - 1} \sum_{l=1}^{N_{ave}} [e^l(t_i) - e^{ave}(t_i)][e^l(t_i) - e^{ave}(t_i)]^T \tag{11}$$

where N_{ave1} is from small sample theory used to obtain an unbiased variance.
We next verify the ensemble average orthogonality of the error by

$$\mathcal{O}^{ave}(t_i) = \frac{1}{N_{ave}} \sum_{l=1}^{N_{ave}} [e^l(t_i) - e^{ave}(t_i)]\hat{x}(t_i)^T \tag{12}$$

And lastly, we verify the independence of the residuals, where the residual is simply the difference from the measurement z to $\bar{z} = H\bar{x}$

$$\begin{aligned} r^l(t_i) &= z(t_i) - H\bar{x} \\ \mathcal{R}^{ave} &= \frac{1}{N_{ave}} \sum_{l=1}^{N_{ave}} r^l(t_i) r^l(t_m)^T \end{aligned} \tag{13}$$

where t_i and t_m are two time values in the simulation run, and $t_m < t_i$.

The validation here for equations (10)-(13) is that they should all be approximately equal to zero: $e^{ave}(t_i) \approx 0, P^{ave}(t_i) \approx 0, \mathcal{O}^{ave}(t_i) \approx 0, \mathcal{R}^{ave}(t_i) \approx 0 \;\forall\; t_i$

2.7 Psuedocode

Here we provide a rundown of the MATLAB algorithm.

1: Initialize all constants and parameters, including Φ, Γ, and H.
2: Initialize arrays to store $\bar{x}$, M, $\hat{x}$, P, K, and $\bar{e}$, as well as $p_c(t)$, $v_c(t)$, and $z(t)$.
3: The arrays are multidimensional in the size of data of the state × time of simulation × number of Monte-Carlo simulation runs.
4: **for** realizations in the Monte-Carlo runs **do**
5: Generate and set the random initial conditions.
6: **for** time steps in the simulation run **do**
7: Generate the process noise and calculate the accelerometer measurements $a_c(t_i)$, $v_c(t_i)$, $p_c(t_i)$.
8: **if** is time for GPS measurement **then**
9: Generate the GPS noise and calculate $\partial p(t_i)$ and $\partial v(t_i)$ to obtain $z(t_i)$.

```
        Perform the Kalman filter update by calculating K(t_i), x̂(t_i), and P(t_i) using equations (4), (5), and (6).
        Perform the Kalman filter propagation by calculating x̄(t_i) and M(t_i) using equations (2) and (3).
      else
        Copy over the previous x̂(t_{i−1}) and P(t_{i−1}).
        Perform the Kalman filter propagation by calculating x̄(t_i) and M(t_i) using the new propagation equations (7) and (8).
      end if
      Calculate the error ē in this time step using equation (9).
    end for
    Alleviate off by one errors in x̂, P, K, and ē by duplicating the previous value.
end for
Calculate P^ave, O^ave, R^ave using equations (11), (12), and (13).
Plot states, variances, and errors.
```

3 Results and Performance

A total of `1000` runs were performed in the Monte-Carlo simulation.

3.1 Single Run Results

From one realization in the Monte-Carlo simulation runs, the truth position, *a priori* position estimate, *posteriori* position estimate, and the accelerometer position estimate are shown below in Figure 1.

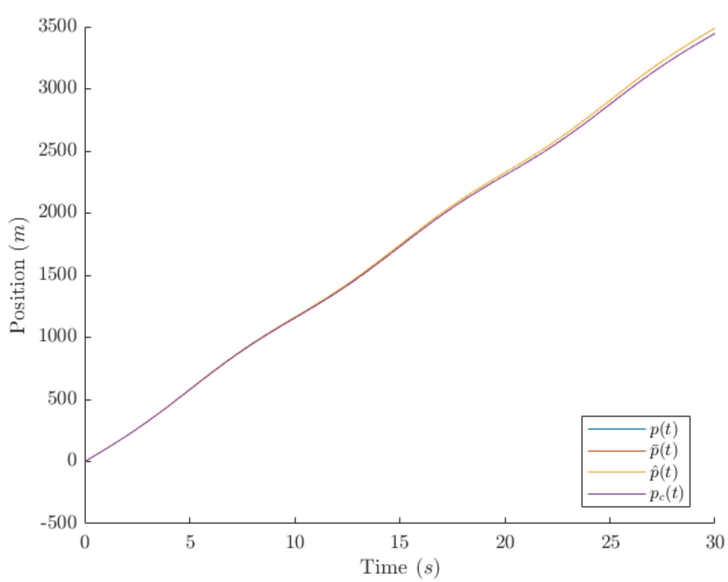


Figure 1: $p(t_i), \bar{p}(t_i), \hat{p}(t_i), p_c(t_i)$

The truth velocity, *a priori* velocity estimate, *posteriori* velocity estimate, and accelerometer velocity estimate are shown below in Figure 2.

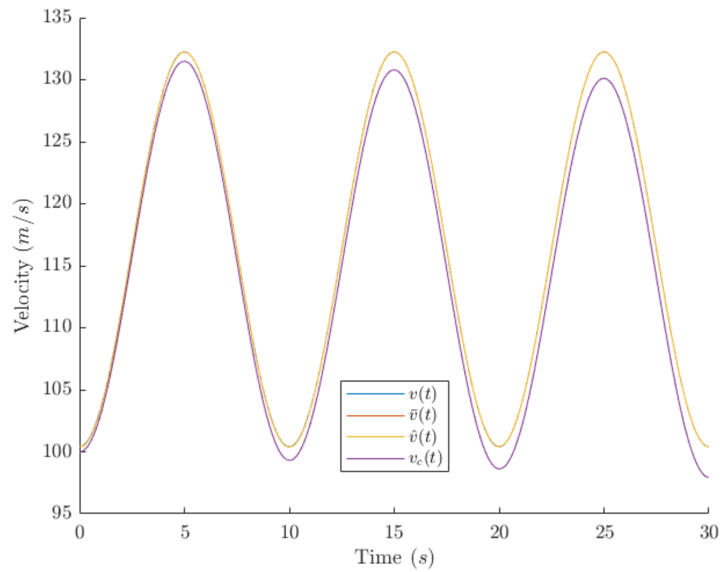


Figure 2: $v(t_i), \bar{v}(t_i), \hat{v}(t_i), v_c(t_i)$

The truth accelerometer bias, *a priori* bias estimate, and *posteriori* bias estimate are shown below in Figure 3.

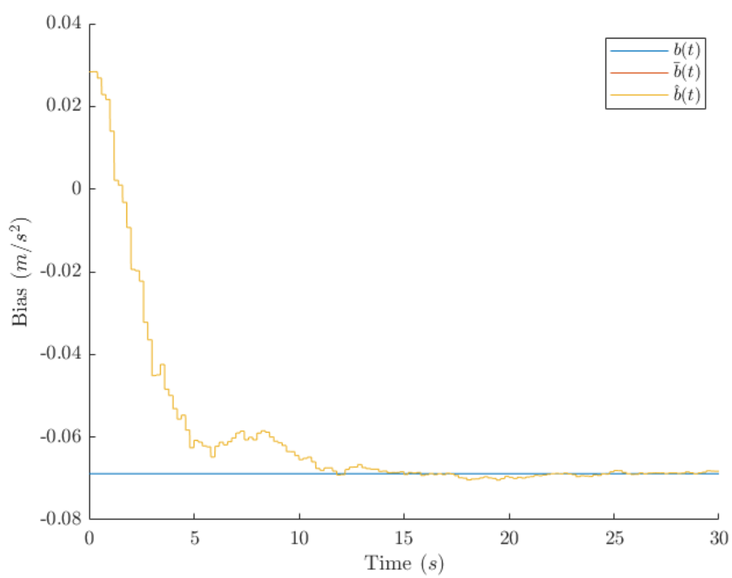


Figure 3: $b(t_i), \bar{b}(t_i), \hat{b}(t_i)$

The truth delta position, *a priori* delta position estimate, and *posteriori* delta position estimate are shown below in Figure 4. Delta indicates the change from the truth model and the accelerometer model.

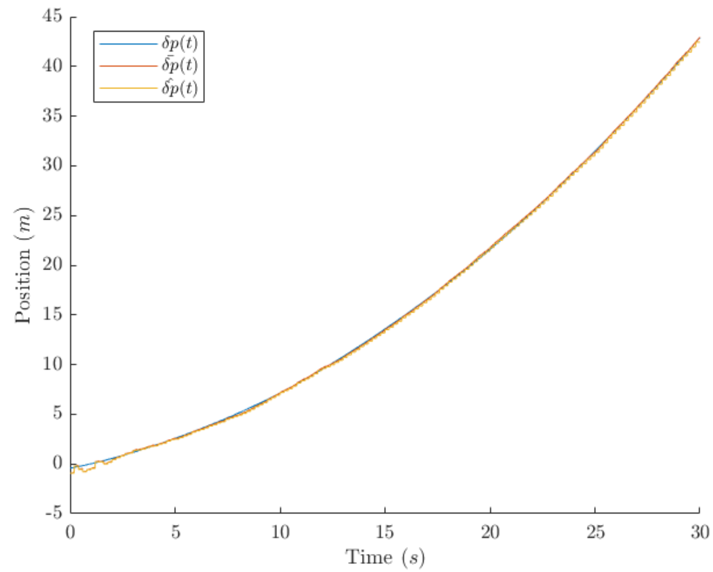


Figure 4: $\partial p(t_i), \partial \bar{p}(t_i), \partial \hat{p}(t_i)$

The truth delta velocity, *a priori* delta velocity estimate, and *posteriori* delta velocity estimate are shown below in Figure 5.

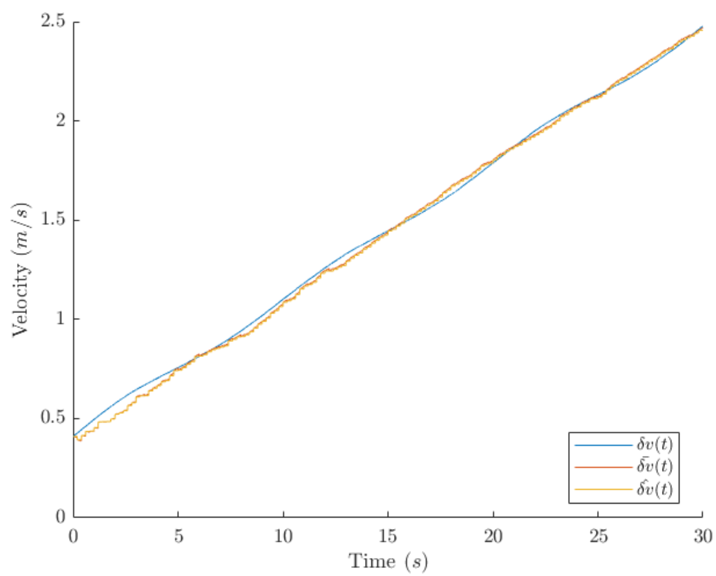


Figure 5: $\partial v(t_i), \partial \bar{v}(t_i), \partial \hat{v}(t_i)$

The *a priori* delta position error estimate, *posteriori* delta position error estimate, *a priori* position standard deviation, and *posteriori* position standard deviation are shown below in Figure 6. Delta error indicates the change from the truth delta and the delta estimate.

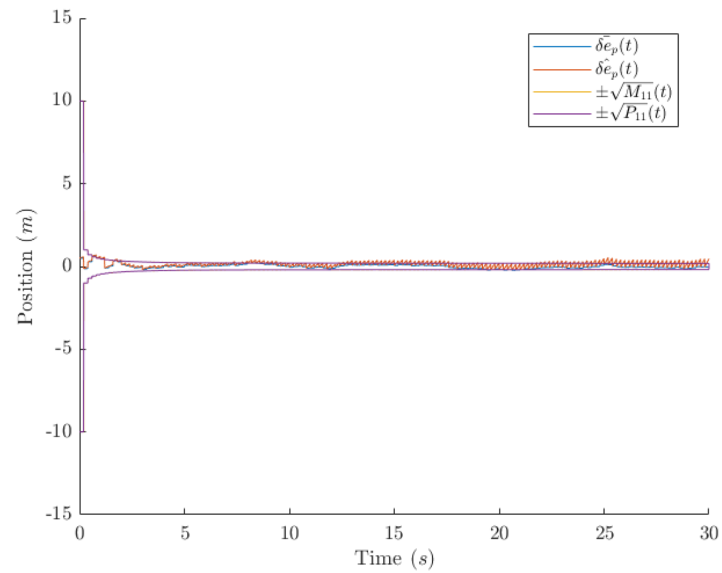


Figure 6: $\partial\bar{e_p}(t_i), \partial\hat{e_p}(t_i), \pm\sqrt{M_{11}}(t_i), \pm\sqrt{P_{11}}(t_i)$

The *a priori* delta velocity error estimate, *posteriori* delta velocity error estimate, *a priori* velocity standard deviation, and *posteriori* velocity standard deviation are shown below in Figure 7.

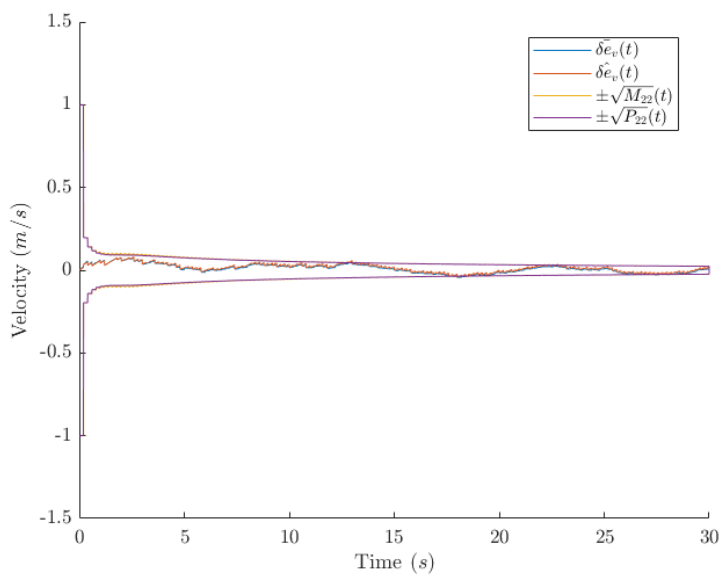


Figure 7: $\partial\bar{e_v}(t_i), \partial\hat{e_v}(t_i), \pm\sqrt{M_{22}}(t_i), \pm\sqrt{P_{22}}(t_i)$

The *a priori* bias error estimate, *posteriori* bias error estimate, *a priori* bias standard deviation, and *posteriori* bias standard deviation are shown below in Figure 8.

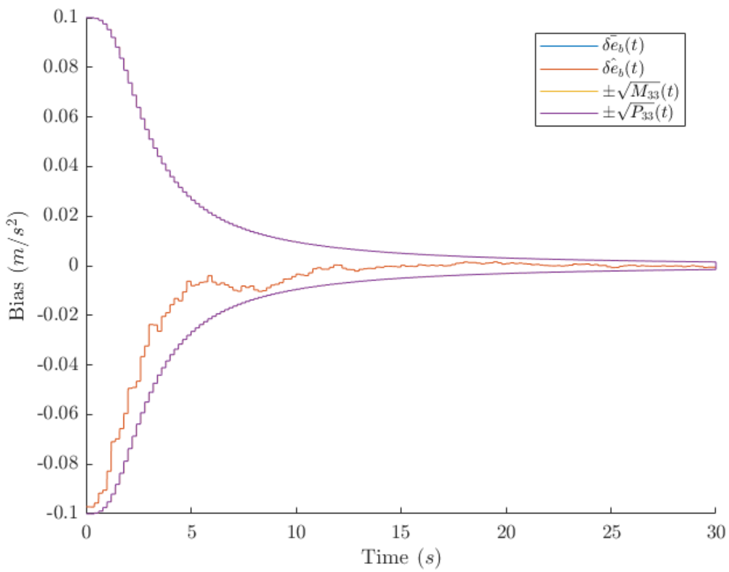
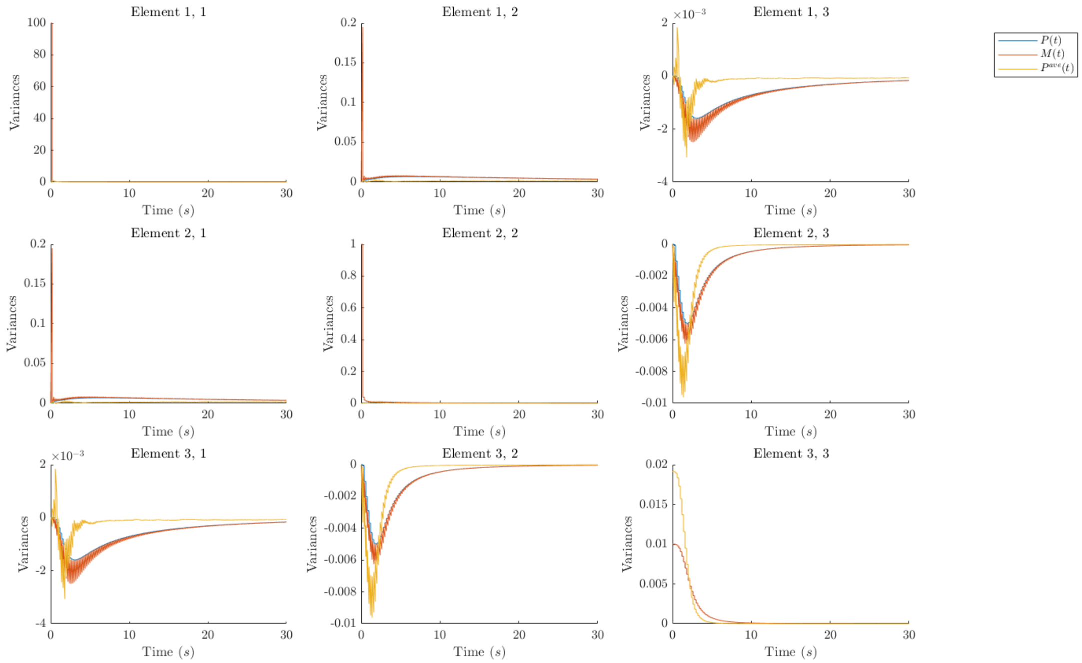


Figure 8: $\bar{e}_b(t_i), \hat{e}_b(t_i), \pm\sqrt{M_{33}}(t_i), \pm\sqrt{P_{33}}(t_i)$

3.2 Ensemble Run Results

Now, across the ensemble of `1000` Monte-Carlo simulations runs, the statistics of the results across all runs are displayed below.

The time history of the *a priori* variances, *posteriori* variances, and average variances across all runs P^{ave}, are shown below in Figure 9.



A larger, more visibly-scaled version below:

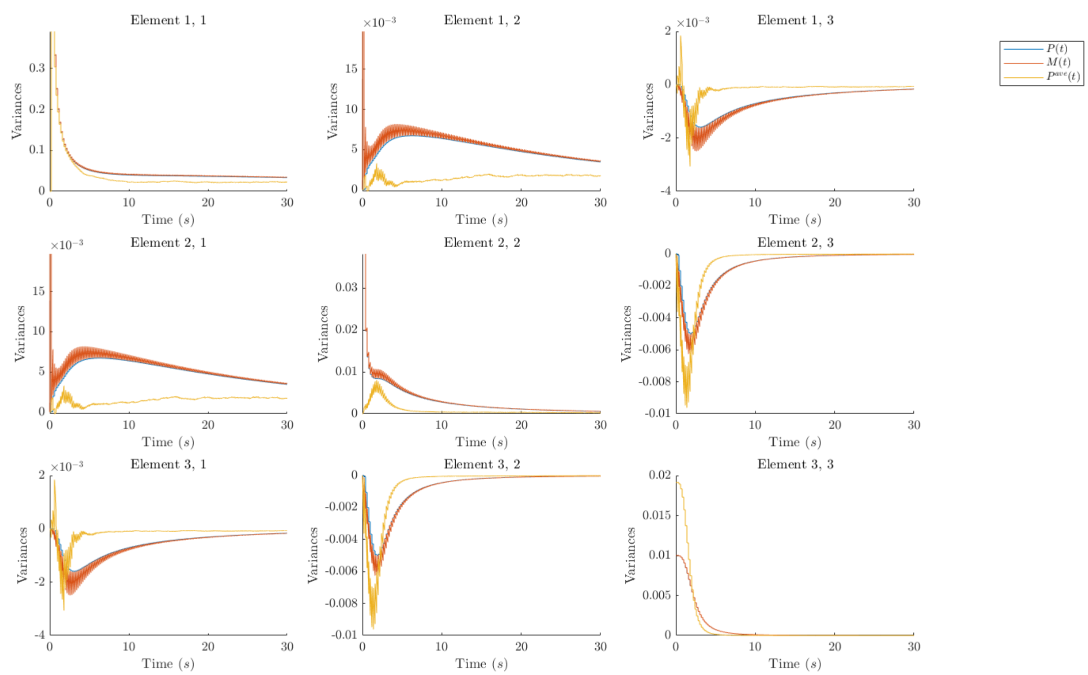
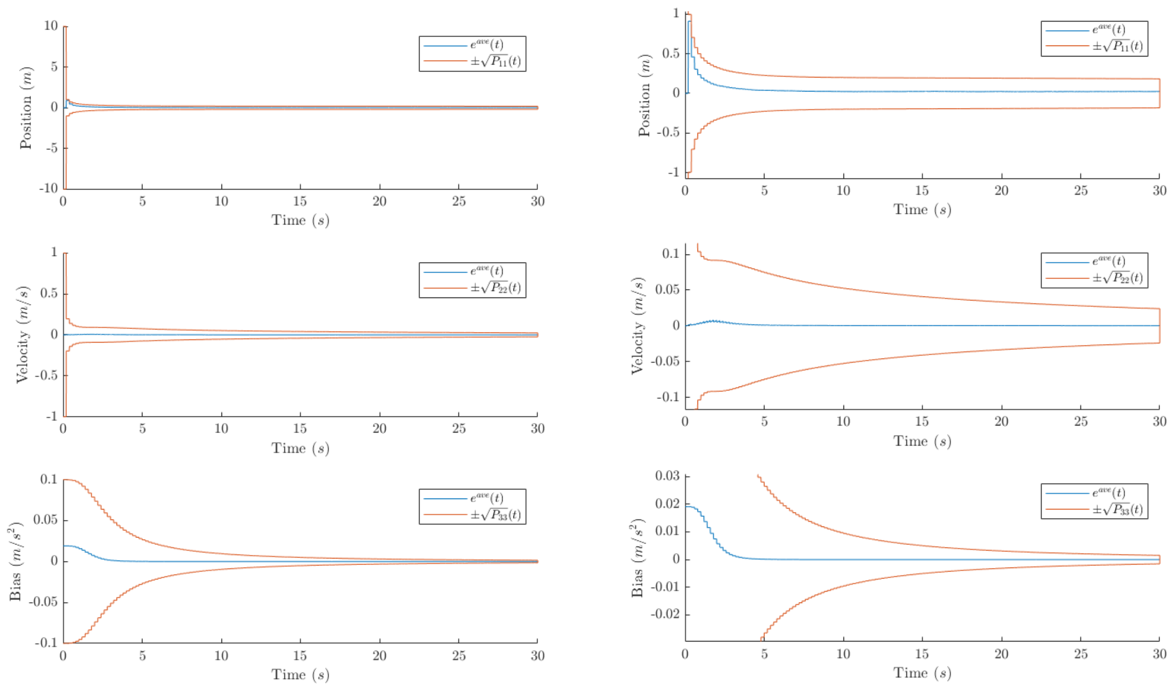


Figure 9: $M(t_i), P(t_i), P^{ave}(t_i)$

The time history of the average error and one σ standard deviation for each of the position, velocity, and bias are shown below in Figure 10.



The plot is duplicated on the right with a larger scale to observe the small features.

Figure 10: $e^{ave}(t_i), \pm\sqrt{P_{ii}^{ave}}(t_i)$

The time history of the orthogonality property $\mathcal{O}^{ave}$ of the simulation is shown below in Figure 11.

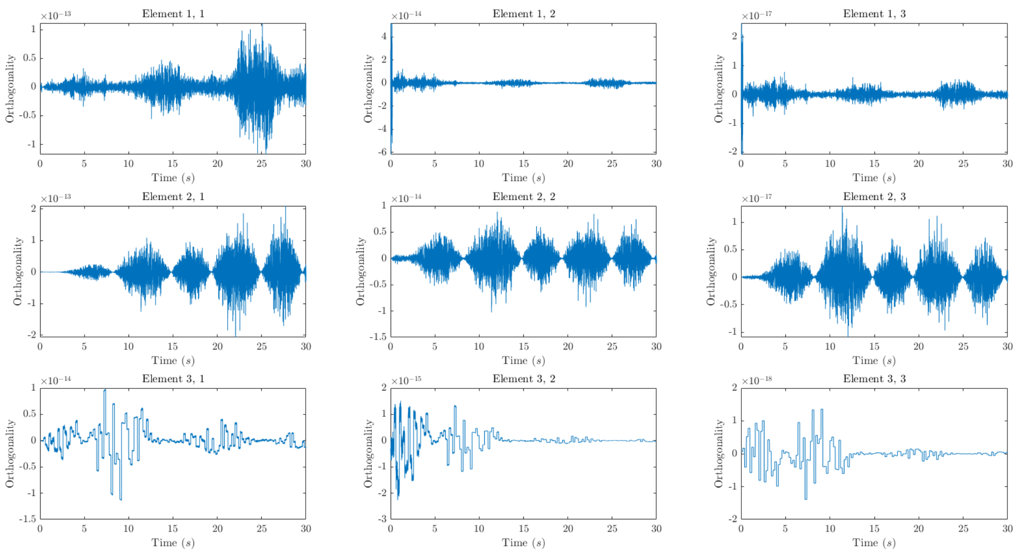


Figure 11: $\mathcal{O}^{ave}(t_i)$

The orthogonal property of the residual $\mathcal{R}^{ave}$ is verified at two time steps t_i and t_m, such that $t_m < t_i$, and calculated using equation (13).

$\mathcal{R}^{ave} =$

```
   0.9131   -0.0011
  -0.0011    0.0016
```

4 Conclusions

The proposed and implemented Kalman filter is observed to work as intended as a minimum variance estimator. We see that in the simulation as the filter runs over time, the estimate states converge to the truth for position, velocity, and the accelerometer bias.

One example can be seen in an enlarged velocity plot of Figure (2).

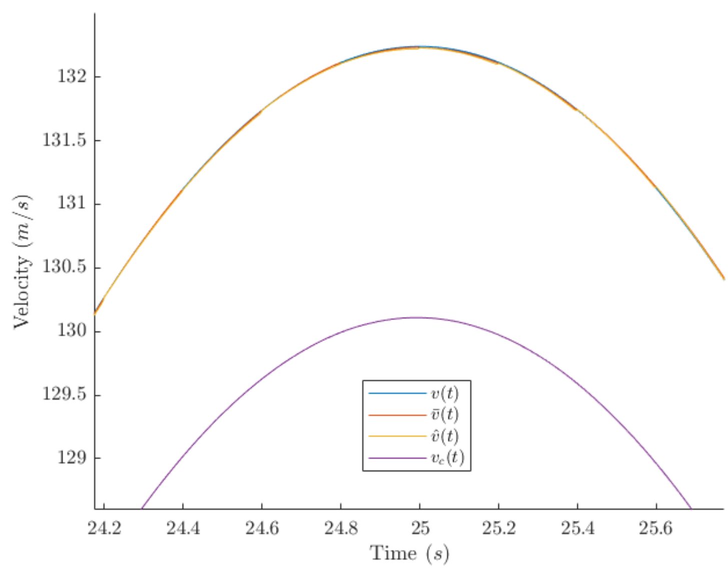


Figure 12: Zoomed in plot of the 3rd peak of Figure(2)

We see the estimates $\bar{v}$ and $\hat{v}$ tracking very closely to the truth v, while the accelerometer measured v_c is the outlier.

We also observe that the estimates reside within the estimated standard deviation $\pm\sqrt{P_i i}$ over time for all states, as seen in Figures 6, 7, and 8. Over time, the standard deviation shrinks as the filter becomes more certain of the estimate, and the error ∂e over time also mostly is bounded by the the standard deviation.

The estimated covariance M, P, and the calculated P^{ave} calculated from e^{ave} all shrink to very small values near 0, as seen in Figure 9, indicating the increasing certainty of the estimate over time. For the covariances (off-diagonal elements of the matrices), we observe sharp oscillations early on in the simulation that shrink as time passes. This is seen as the covariance growing very rapidly between measurement updates. The growth slows as P decreases with more measurements.

We observe that the orthogonality properties are upheld in that all the values of $\mathcal{O}^{ave}$ are very small, as seen in Figure 11, where the largest errors are of magnitude 10^{-13}.

Hence, this implementation of the discrete Kalman filter is capable of estimating the states and accelerometer bias of the vehicle using noisy accelerometer and GPS measurements.

5 Code Listing

```
% James Tseng
% MAEC175A Final Project
clear; close all; clc;

%% Parameters
% Simulation
n_run = 1000;           % monte carlo
p_run = 222;            % select which run to plot
t_run = 30;             % s
dt = 0.005;             % s     time step
s_size = t_run/dt;      % history size
dt_run = dt:dt:t_run;   % array of time

% Truth
w = 2*pi*0.1;           % rad/s
aa = 10;                % amplitude
a = @(t) aa*sin(w*t);

% Accelerometer
a_samp = 200;           % Hz
a_w_mean = 0;           % additive white Guassian noise, 0 mean
a_w_vari = 0.0004;      % (m/s^2)^2     variance
% a_b bias               % varies with run
a_b_mean = 0;
a_b_vari = 0.01;        % (m/s^2)^2

% GPS
g_samp = 5;             % Hz
x0_g_mean = 0;          % m     a priori starting position
x0_g_vari = 100;        % m^2
v0_g_mean = 100;        % m/s
v0_g_vari = 1;          % (m/s)^2
np_g_mean = 0;          % additive white noise
np_g_vari = 1;          % m^2
nv_g_mean = 0;
nv_g_vari = (4/100)^2;      % (m/s)^2
n_g_vari = [1 0; 0 (4/100)^2];

%% Simulation Setup
% Kalman Filter 4 Equations
% system dynamics constant
Phi = [1 dt -dt^2/2; 0 1 -dt; 0 0 1];
```

```
Gam = [dt^2/2; dt; 0];
H = [1 0 0; 0 1 0];
% store all history across all runs
dx_bar = zeros(3,1,s_size,n_run);
M = zeros(3,3,s_size,n_run);
dx_hat = zeros(3,1,s_size,n_run);
P = zeros(3,3,s_size,n_run);
K = zeros(3,2,s_size,n_run);
e_bar = zeros(3,1,s_size,n_run);
% accelerometer model
vc = zeros(s_size,n_run);
pc = zeros(s_size,n_run);
b = zeros(n_run,1);
% gps model
dz = zeros(2,1,s_size,n_run);
% truth models
p = cell(n_run,1);
v = cell(n_run,1);

%% Monte Carlo Loop
for l = 1:n_run         % realization l
    % for debug
    if mod(l,100) == 0
        disp(l);
    end
    % initial states
    % truth
    v0 = random('Normal',v0_g_mean,sqrt(v0_g_vari));
    p0 = random('Normal',x0_g_mean,sqrt(x0_g_vari));
    v{l} = @(t) v0 + aa/w - aa/w*cos(w*t);
    p{l} = @(t) p0 + (v0 + aa/w)*t - aa/w^2*sin(w*t);
    % accelerometer combined with dynamics model since dt = a_samp
    pc0 = x0_g_mean;
    vc0 = v0_g_mean;
    b(l) = random('Normal',a_b_mean,sqrt(a_b_vari));
    b0 = random('Normal',a_b_mean,sqrt(a_b_vari));
    % set initial conditions
    dx_hat(:,1,1,l) = [p0-pc0; v0-vc0; b0];
    dx_bar(:,1,1,l) = dx_hat(:,1,1,l);
    vc(1,l) = vc0;
    pc(1,l) = pc0;
    M(:,:,1,l) = diag([x0_g_vari v0_g_vari a_b_vari]);
    P(:,:,1,l) = M(:,:,1,l);

```

```
    %% Simulation Loop
    for i = 1:1:s_size-1
        t = i*dt;

        % accelerometer model
        wpn = random('Normal',a_w_mean,sqrt(a_w_vari)); % process noise
        ac = a(t) + b(l) + wpn;
        vc(i+1,l) = vc(i,l) + ac*dt;
        pc(i+1,l) = pc(i,l) + vc(i,l)*dt + ac*dt^2/2;

        % update if GPS has measurement
        if mod(t,1/g_samp) == 0
            % get measurement
            dp = p{l}(t) - pc(i,l); dv = v{l}(t) - vc(i,l);
            np = random('Normal',np_g_mean,sqrt(np_g_vari));
            nv = random('Normal',nv_g_mean,sqrt(nv_g_vari));
            dz(:,1,i,l) = [dp + np; dv + nv];
            % update
            K(:,:,i,l) = M(:,:,i,l)*H.'/(H*M(:,:,i,l)*H.' + n_g_vari);
            dx_hat(:,1,i,l) = dx_bar(:,1,i,l) + K(:,:,i,l)*(dz(:,1,i,l) - H*dx_bar(:,1,i,l));
            dx_bar(:,1,i,l) = dx_hat(:,1,i,l);
            P(:,:,i,l) = M(:,:,i,l) - K(:,:,i,l)*H*M(:,:,i,l);
            % propagate
            dx_bar(:,1,i+1,l) = Phi*dx_hat(:,1,i,l);
            M(:,:,i+1,l) = Phi*P(:,:,i,l)*Phi.' + Gam*a_w_vari*Gam.'; % since wpn scalar, cov = var

        else
            if i>1 % copy over previous P and dx_hat
                K(:,:,i,l) = K(:,:,i-1,l);
                P(:,:,i,l) = P(:,:,i-1,l);
                dx_hat(:,1,i,l) = dx_hat(:,1,i-1,l);
            end
            % propagate using predicted dx_bar and M
            dx_bar(:,1,i+1,l) = Phi*dx_bar(:,1,i,l);
            M(:,:,i+1,l) = Phi*M(:,:,i,l)*Phi.' + Gam*a_w_vari*Gam.';
        end

        % measure error
        e_bar(:,1,i,l) = [p{l}(t) - pc(i,l); v{l}(t) - vc(i,l); b(l)] - dx_bar(:,1,i,l);
    end

```

```
    % copy over last P and dx_hat at final t
    K(:,:,s_size,l) = K(:,:,s_size-1,l);
    P(:,:,s_size,l) = P(:,:,s_size-1,l);
    dx_hat(:,1,s_size,l) = dx_hat(:,1,s_size-1,l);
    e_bar(:,1,s_size,l) = [p{l}(t_run) - pc(s_size,l); v{l}(t_run) - vc(s_size,l); b(l)] - dx_bar(:,1,s_size,l);
end

%% Plots from Single 30s Run
set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');

%% p, p_bar, p_hat, pc
p_t = arrayfun(p{p_run},dt_run);
figure(1);
hold on;
plot(dt_run,p_t,'DisplayName','$p(t)$');
plot(dt_run,squeeze(dx_bar(1,1,:,p_run))+pc(:,p_run), 'DisplayName', '$\bar{p}(t)$');
plot(dt_run,squeeze(dx_hat(1,1,:,p_run))+pc(:,p_run), 'DisplayName', '$\hat{p}(t)$');
plot(dt_run,pc(:,p_run),'DisplayName','$p_c(t)$');
xlabel('Time ($s$)'); ylabel('Position ($m$)');
legend('Location', 'Best')

%% v, v_bar, v_hat, vc
v_t = arrayfun(v{p_run},dt_run);
figure(2);
hold on;
plot(dt_run,v_t,'DisplayName','$v(t)$');
plot(dt_run,squeeze(dx_bar(2,1,:,p_run))+vc(:,p_run), 'DisplayName', '$\bar{v}(t)$');
plot(dt_run,squeeze(dx_hat(2,1,:,p_run))+vc(:,p_run), 'DisplayName', '$\hat{v}(t)$');
plot(dt_run,vc(:,p_run),'DisplayName','$v_c(t)$');
xlabel('Time ($s$)'); ylabel('Velocity ($m/s$)');
legend('Location', 'Best')

%% b, b_bar, b_hat
figure(3);
hold on;
plot(dt_run,b(p_run)*ones(1,s_size),'DisplayName','$b(t)$');
plot(dt_run,squeeze(dx_bar(3,1,:,p_run)), 'DisplayName', '$\bar{b}(t)$');
```

```
plot(dt_run,squeeze(dx_hat(3,1,:,p_run)), 'DisplayName', '$\hat{b}(t)$');
xlabel('Time ($s$)'); ylabel('Bias ($m/s^2$)');
legend('Location', 'Best')

%% dp, dp_bar, dp_hat
dp_t = p_t-pc(:,p_run).';
figure(4);
hold on;
plot(dt_run,dp_t,'DisplayName','$\delta p(t)$');
plot(dt_run,squeeze(dx_bar(1,1,:,p_run)), 'DisplayName', '$\bar{\delta p}(t)$');
plot(dt_run,squeeze(dx_hat(1,1,:,p_run)), 'DisplayName', '$\hat{\delta p}(t)$');
xlabel('Time ($s$)'); ylabel('Position ($m$)');
legend('Location', 'Best')

%% dv, dv_bar, dv_hat
dv_t = v_t-vc(:,p_run).';
figure(5);
hold on;
plot(dt_run,dv_t,'DisplayName','$\delta v(t)$');
plot(dt_run,squeeze(dx_bar(2,1,:,p_run)), 'DisplayName', '$\bar{\delta v}(t)$');
plot(dt_run,squeeze(dx_hat(2,1,:,p_run)), 'DisplayName', '$\hat{\delta v}(t)$');
xlabel('Time ($s$)'); ylabel('Velocity ($m/s$)');
legend('Location', 'Best')

%% de_p_bar, de_p_bar, +-sqrt(M_11), +-sqrt(P_11)
M_11 = squeeze(sqrt(M(1,1,:,p_run)));
P_11 = squeeze(sqrt(P(1,1,:,p_run)));
figure(6);
hold on;
plot(dt_run,dp_t-squeeze(dx_bar(1,1,:,p_run)).', 'DisplayName', '$\bar{\delta e_p}(t)$');
plot(dt_run,dp_t-squeeze(dx_hat(1,1,:,p_run)).', 'DisplayName', '$\hat{\delta e_p}(t)$');
plot([dt_run,fliplr(dt_run)],[M_11;flipud(-M_11)], 'DisplayName', '$\pm\sqrt{M_{11}}(t)$');
plot([dt_run,fliplr(dt_run)],[P_11;flipud(-P_11)], 'DisplayName', '$\pm\sqrt{P_{11}}(t)$');
xlabel('Time ($s$)'); ylabel('Position ($m$)');
legend('Location', 'Best')
```

```
%% de_v_bar, de_v_bar, +-sqrt(M_22), +-sqrt(P_22)
M_22 = squeeze(sqrt(M(2,2,:,p_run)));
P_22 = squeeze(sqrt(P(2,2,:,p_run)));
figure(7);
hold on;
plot(dt_run,dv_t-squeeze(dx_bar(2,1,:,p_run)).', 'DisplayName', '$\bar{\delta e_v}(t)$');
plot(dt_run,dv_t-squeeze(dx_hat(2,1,:,p_run)).', 'DisplayName', '$\hat{\delta e_v}(t)$');
plot([dt_run,fliplr(dt_run)],[M_22;flipud(-M_22)], 'DisplayName', '$\pm\sqrt{M_{22}}(t)$');
plot([dt_run,fliplr(dt_run)],[P_22;flipud(-P_22)], 'DisplayName', '$\pm\sqrt{P_{22}}(t)$');
xlabel('Time ($s$)'); ylabel('Velocity ($m/s$)');
legend('Location', 'Best')

%% de_b_bar, de_b_bar, +-sqrt(M_33), +-sqrt(P_33)
M_33 = squeeze(sqrt(M(3,3,:,p_run)));
P_33 = squeeze(sqrt(P(3,3,:,p_run)));
figure(8);
hold on;
plot(dt_run,b(p_run)*ones(s_size,1)-squeeze(dx_bar(3,1,:,p_run)), 'DisplayName', '$\bar{\delta e_b}(t)$');
plot(dt_run,b(p_run)*ones(s_size,1)-squeeze(dx_hat(3,1,:,p_run)), 'DisplayName', '$\hat{\delta e_b}(t)$');
plot([dt_run,fliplr(dt_run)],[M_33;flipud(-M_33)], 'DisplayName', '$\pm\sqrt{M_{33}}(t)$');
plot([dt_run,fliplr(dt_run)],[P_33;flipud(-P_33)], 'DisplayName', '$\pm\sqrt{P_{33}}(t)$');
xlabel('Time ($s$)'); ylabel('Bias ($m/s^2$)');
legend('Location', 'Best')

%% M, P, P^ave
e_ave = 1/n_run*sum(e_bar,4);
P_ave = zeros(3,3,s_size);
for i=1:1:s_size
    for l=1:1:n_run
    P_ave(:,:,i) = P_ave(:,:,i) + (e_bar(:,:,i,l)-e_ave(:,:,i))*...
                                  (e_bar(:,:,i,l)-e_ave(:,:,i)).';
    end
end
P_ave = P_ave/(n_run-1);
% plot just one M and P since all should be the same
figure(9);
```

```
k = 0;
for i = 1:1:3
    for j = 1:1:3
            subplot(3,4,3*(i-1)+j+k); hold on;
            plot(dt_run,squeeze(P(i,j,:,p_run)));
            plot(dt_run,squeeze(M(i,j,:,p_run)));
            plot(dt_run,squeeze(P_ave(i,j,:)));
            xlabel('Time ($s$)'); ylabel('Variances');
            title(['Element ',num2str(i),', ',num2str(j)]);
    end
    k = k+1; % deal with 3x4 subplot
end
subplot(3,4,4)
plot(0,0,  0,0,  0,0,  0,0)
axis off
legend('$P(t)$','$M(t)$','$P^{ave}(t)$','Location','northwest')

%% e^ave, +-sqrt(P_ii) for 1<=ii<=3
figure(10);
subplot(3,1,1); hold on;
plot(dt_run,squeeze(P_ave(1,1,:)),'DisplayName','$e^{ave}(t)$');
plot([dt_run,fliplr(dt_run)],[P_11;flipud(-P_11)], 'DisplayName',
↪  '$\pm\sqrt{P_{11}}(t)$');
xlabel('Time ($s$)'); ylabel('Position ($m$)'); legend
subplot(3,1,2); hold on;
plot(dt_run,squeeze(P_ave(2,2,:)),'DisplayName','$e^{ave}(t)$');
plot([dt_run,fliplr(dt_run)],[P_22;flipud(-P_22)], 'DisplayName',
↪  '$\pm\sqrt{P_{22}}(t)$');
xlabel('Time ($s$)'); ylabel('Velocity ($m/s$)'); legend
subplot(3,1,3); hold on;
plot(dt_run,squeeze(P_ave(3,3,:)),'DisplayName','$e^{ave}(t)$');
plot([dt_run,fliplr(dt_run)],[P_33;flipud(-P_33)], 'DisplayName',
↪  '$\pm\sqrt{P_{33}}(t)$');
xlabel('Time ($s$)'); ylabel('Bias ($m/s^2$)'); legend

%% Orthogonality Property of Simulation
x_t(1,1,:) = p_t;
x_t(2,1,:) = v_t;
x_t(3,1,:) = b(p_run)*ones(1,s_size);
x_hat = dx_hat(:,:,:,p_run) + x_t;
orth = zeros(3,3,s_size);
for i=1:1:s_size
    for l=1:1:n_run
    orth(:,:,i) = orth(:,:,i) +
↪  (e_bar(:,:,i,l)-e_ave(:,:,i))*x_hat(:,:,i).';
```

```
    end
end
orth = orth/n_run;
figure(11);
for i = 1:1:3
    for j = 1:1:3
            subplot(3,3,3*(i-1)+j);
            plot(dt_run,squeeze(orth(i,j,:)));
            xlabel('Time ($s$)'); ylabel('Orthogonality');
            title(['Element ',num2str(i),', ',num2str(j)]);
    end
end

%% Orthogonality Property of Residual
ti = p_run/g_samp/dt; ti = min([ti, s_size-1/g_samp/dt]); tm = ti/2;
corr_res = zeros(2,2);
for l=1:1:n_run
    rti = dz(:,1,ti,l) - H*dx_bar(:,1,ti,l);
    rtm = dz(:,1,tm,l) - H*dx_bar(:,1,tm,l);
    corr_res = corr_res + squeeze(rti)*squeeze(rti).';
end
corr_res = corr_res/n_run
```