UCLA Samueli School of Engineering
Mechanical and Aerospace Engineering Department
University of California, Los Angeles

Probability and Stochastic Processes (271A)
December 1, 2022

Project: Calibration of an Accelerometer using GPS Measurements

Student: Arturo Flores Alvarez
Email: afloresa@ucla.edu

Contents

1 Abstract

Odometry is the process of using data from motion sensors such as Inertial Measurement Units (IMUs) or GPS devices in order to estimate changes in position over time. This concept is widely applied in the field of robotics to address the problem of navigation and traversability in various terrain. More specifically, in underground, GPS-denied environments, a simpler navigation system is normally taken to surpass the constraints of communication and power of such places. Therefore, having a good estimation from an IMU is critical to reaching the objectives on the field for a robot. Unfortunately, while these electronic components are very cheap and handy for everyday applications, their measurements are corrupted by stochastic processes such as noise and bias that motivate their frequent calibration. In this manuscript, a method to calibrate a 1-D accelerometer using a GPS through a Kalman Filter is proposed. Moreover, the accelerometer's position and velocity are estimated through its kinematic equations and compared with those measured by the GPS in an interval of t=30s. The results show that the bias is properly estimated, and through Monte Carlo experiments over different epochs, the estimator can reduce the average error and variance in all the system's variables of interest. This didactic example shows step-by-step the correct implementation of a minimum variance estimator and proves the theoretical orthogonality properties of stochastic processes.

2 Introduction

2.1 Problem Statement

Initially, we are provided with the current/true acceleration profile in one dimension of a vehicle with an accelerometer

$$a(t) = A \cdot \sin(\omega t) \;\; m/sec^2 \tag{1}$$

Since these electronic devices are prone to introduce error to their measurements, the accelerometer measurements are modeled with an additive Gaussian noise w and a bias b_a, both with known a priori statistics.

$$a_c(t_j) = a(t_j) + b_a + w(t_j) \tag{2}$$

On the other hand, the GPS is used to provide measured position and velocity synchronized with the accelerometer. Likewise, the GPS measurements are modeled with noises sequences with a priori known statistics in the following way

$$\begin{aligned} z_{1_i} &= x(t_i) + \eta_{1_i} \\ z_{2_i} &= v(t_i) + \eta_{2_i} \end{aligned} \tag{3}$$

Table 1 and 2 summarize all the information provided by the prompt of the assignment.

Table 1: Summary of all Accelerometer parameters provided by the problem

#	Parameter	Variable	Value	Unit
	Accelerometer			
1	Sampling Rate	f_{IMU}	200	Hz
2	Amplitude	A	10	m/s^2
3	Experiment Length	T	30	s
4	Frequency	ω	0.1	rad/s
5	Bias Mean	$\overline{b}_0$	0	m/s^2
6	Noise bias	$\overline{w}_0$	0	m/s^2
7	Bias Variance	σ_b^2	0.01	$(m/s^2)^2$
8	Noise Variance	σ_w^2	0.0004	$(m/s^2)^2$

Table 2: Summary of all GPS parameters provided by the problem

#	Parameter	Variable	Value	Unit
	GPS-Acceleration			
9	Sampling Rate	f_{gps}	5	Hz
3	Experiment Length	T	30	s
	GPS-Pos- A priori			
10	Position Mean	$\overline{x}$	0	m
11	Position Variance	${\sigma^2}_x$	100	m^2
12	Position Noise Mean	η_{p0}	0	m
13	Position Noise Variance	${\sigma^2}_{\eta,p0}$	1	m^2
	GPS-Vel- A priori			
14	Velocity Mean	$\overline{v}$	0	(m/s)
15	Velocity Variance	${\sigma^2}_v$	1	$(m/s)^2$
16	Velocity Noise Mean	η_{v0}	0	(m/s)
17	Velocity Noise Variance	${\sigma^2}_{\eta,v0}$	0.0016	$(m/s)^2$

It is worth highlighting that the initial conditions for the position and velocity in Equation (3) are given by variables 10 and 12 in Table (2).

2.2 Objectives

In this manuscript, the first section recaps all the information provided by the problem statement and the objectives of the assignment. The Section 'Theory' essentially introduces concepts related to the Stochastic Processes Theory used for the design of the Kalman filter. Next, the section 'Methods & Results' goes over step-by-step the algorithm designed in MATLAB for the Kalman Filter and presents the plots obtained for each section. Finally, the last Section 'Conclusions' summarizes the important results of this work. According to the assignment prompt, these are expectations for the work

1. Determine an associated stochastic discrete-time system that is approximately independent of the acceleration profile.

2. Design a Kalman filter and plot the estimates of position, velocity, an accelerometer bias as well as the filter error variance (considering one sigma bound)

3. For the validation of the minimum variance estimator, we should follow a Monte Carlo approach to do ensemble realizations. Specifically, the batches proposed are [10 40 160 640] realizations

    - Proof that over an ensemble of realizations, the simulated error variance is close to the filter error variance
    - Show the theoretical orthogonalities and independence properties are satisfied for this filter.

3 Theory

3.1 Inertial Measurement Unit

An Inertial Measurement Unit, also known as IMU, is an electronic device that measures and reports acceleration, orientation, angular rates, and other gravitational forces. It is normally composed of 3 accelerometers, 3 gyroscopes, and depending on the heading requirement – 3 magnetometers [3].

Figure 1: Comercial Inertial Measurement Unit. Figure extracted from [3]

3.2 Conditional Probability

The conditional probability of event B is the probability that the event will occur given the knowledge that event A has already occurred. This probability is written $P(B|A)$, a notation for the probability of B given A. In the case where events A and B are independent (where event A has no effect on the probability of event B), the conditional probability of event B given event A is simply the probability of event B, that is P(B) [4].

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \tag{4}$$

The conditional probability density function of X, given that $Y = y$, is defined by [2]:

$$g(x|y) = \frac{f(x,y)}{f_Y(y)} \tag{5}$$

In the proposition above, we assume that the marginal pmf $f_Y(y)$ is known. If it is not, it can be derived from the joint $f_{XY}(x,y)$ by marginalization.

3.3 Gauss-Markov Processes

A random variable is a Markov process when the future is independent of the past. Therefore, For all $t > s$ and arbitrary values $x(t), x(s)$ and $x(u)$ for all $u < s$, it can be affirmed that

$$P(X(t) \leq x(t))|X(s) \leq x(s), X(u) \leq x(u)) = P(X(t) \leq x(t))|X(s) \leq x(s)) \tag{6}$$

We can say that this process is memoryless because the future doesn't depend on previous instants. On the other hand, a process is said to be Gaussian when all probability distributions are Gaussian [1]. For arbitrary $n > 0$, the sample instants $t_1, t_2, ..., t_n$ their corresponding realizations

$$X(t_1), X(t_2), \ldots X(t_n) \tag{7}$$

are jointly Gaussian Random variables. Moreover, these random variables should be related to either a sum or a scalar multiplication to keep the Gaussian condition. A Gauss-Markov process preserves both Gaussian and Markov conditions.

3.4 Discrete Kalman Filter

The continuous estimation of a set of parameters whose values change over time is the problem we are seeking to solve. For this purpose, the conditional probability density function $f_{x|z}$ contains all the minimum information that we need to solve any estimation problem. Thus, we want to estimate the current state of our system x given the measurements we have of it z, and to minimize the minimum variance of the estimate of a state $\mathbf{x}$. The mathematical formulation is the following

$$\tilde{\mathbf{x}} = \underset{\tilde{\mathbf{x}}}{\operatorname{argmin}} E[||\tilde{\mathbf{x}} - x||^2|z] = E[x|z]$$

In this case, the conditional mean will be the solution to the minimization problem. The system we want to minimize has the formulation below:

$$\begin{aligned} x_{k+1} &= \Phi_k x_k + \Gamma_k W_k \\ z_x &= H_k x_k + v_k \end{aligned} \tag{8}$$

And we are going to have two stages for the Kalman filter, First, we will propagate the error in the system and then update it with the computed estimation for each measured instant. This, the a priori state mean to the formulation for the propagation stage

$$\begin{aligned} \bar{x}_{k+1} &= \Phi_k \hat{x}_k \\ M_{k+1} &= \Phi_k P_k \Phi_k^T + \Gamma_k W_k \Gamma_k^T \end{aligned} \tag{9}$$

Where the apriori error variance matrix in the next instant M_{k+1} is calculated based on the a posteriori error variance matrix in the previous instant P_k. Then, for the update stage

$$\begin{aligned} \hat{x}_k &= \bar{x}_k + P_k H_k^T V_k^{-1}(z_k - H_k \bar{x}_k) \\ P_k &= \Big(M_k^{-1} + H_k^T V_k^{-1} H_k\Big)^{-1} \end{aligned} \tag{10}$$

Considering that P_k can reach values near zero and be singular, we can use the matrix inversion Lemma to rewrite the P_k matrix into a more computationally efficient fashion

$$P_k = \Big(M_k^{-1} + H_k^T V_k^{-1} H_k\Big)^{-1}$$

$$P_k = \Big(I - K_k H_k\Big) M_k \Big(I - K_k H_k\Big)^T + K_k V_k K_k^T$$

(11)

Furthermore, we can define two additional variables called residual r_k, and the Kalman gain K_k, defined in the following way

$$\begin{aligned} r_k &= z_k - H_k \bar{x}_k \\ K_k &= P_k H_k^T V_k^{-1} \end{aligned} \tag{12}$$

Hence, the updating process for $\hat{x}$ can reduce the Equation 10 can be understood as operating the filter's gain upon the residual and it is presented below

$$\hat{x}_k = \bar{x}_k + P_k H_k^T V_k^{-1}(z_k - H_k \bar{x}_k) \longrightarrow \hat{x}_k = \bar{x}_k + K_k r_k \tag{13}$$

Moreover, errors can be included in the system formulation 8 by defining a new state vector. Thus the a priori and a posteriori errors can be defined like this

$$\begin{aligned} \bar{e}_k &= x_k - \bar{x}_k \\ &= \Phi_{k-1}\bar{e}_{k-1} + \Gamma_{k-1}w_{k-1} \\ e_k &= x_k - \hat{x}_k \\ &= \Phi_{k-1}e_{k-1} + \Gamma_{k-1}w_{k-1} \end{aligned} \tag{14}$$

Where the a posteriori error is orthogonal to the estimated state and therefore is independent. Mathematically, this property is expressed as follows

$$E[e_k\hat{x}]_k^T = 0 \tag{15}$$

On the other hand, the residuals are an independent sequence of all the previous measurements as well, then the innovation process referred to is presented below,

$$E[r_k r_j^T] = 0 \quad \forall j < k \tag{16}$$

These equations will be implemented in an algorithm in the next Section using MATLAB 2019b

4 Methods & Results

In this section, the algorithm developed for the solution of the assignment is presented and explained. Next, the key plots of each intermediate result are included.

4.1 Algorithm

In this section, it will be explained code excerpts from the main code. The excerpts that start with a function are located a the end of the main MATLAB file. The full code for this Kalman Filter is provided in the Appendix,

4.1.1 Initialization

This code represents the variable initialization for each of the values in Table 1 and 2.

```
%% 1. Variable  Initialization
clc
clear  all
% 1.A Accelerometer / Data from the problem
s_rate  = 200;        % 1/s,    Variable 1
step  =(1/s_rate);
Ampl = 10;            % m/s^2, Variable 2
L = 30 ;              % s, Experiment length,Variable 3
w = 0.1   ;           % f,  2pi,Variable 4

% 1.B Accelerometer Truth
t_sin  = [0:(step):L];
acc_sin  = Ampl*sin(w*t_sin);
n_samples = length(t_sin);

% 1.C Accelerometer Measured: Bias + Noise
bias_mean = 0;        %(m/s^2)^(2),Variable 5
noise_mean = 0;       %(m/s^2)^(2),Variable 6

```

```
bias_var = 0.01;      %(m/s^2)^(2),Variable 7
noise_var = 0.0004; %(m/s^2)^(2),Variable 8

% 1.D GPS, 'Current' Dynamics: Position & Velocity
% prior position
freq_gps = 5;         %Hz, synchronized with accelerometer,Variable 9
p_mean = 0;           % 0 meters,Variable 10
p_var = 100;          % (10 meters)^2,Variable 11
% prior velocity
v_mean = 100;         % 10 meters/s,Variable 14
v_var = 1;            % (1 meters/s)^(2),Variable 15

%A priori stadistics for the dynamic noise processes
etap_mean = 0;          %m,Variable 12
etap_var = 1;           %m^2,Variable 13
%velocity noise
etav_mean = 0;          %cm/m,Variable 16
etav_var = (0.04)^2;  %convert(4cm/s)^2 to (m/s)^2,,Variable 17
```

4.1.2 Accelerometer model

Then, the current and measured acceleration, velocity, and position are computed from the current/true acceleration that has the sine function profile provided at the beginning.

```
%% 2. Accelerometer
[acc_measu, bias_model, noise_model] = acc_mod(acc_sin, noise_mean, noise_var, bias_mean, bias_var,n_samples);
%IDEAL+BIAS+NOISE=REAL
[pos_mod, vel_mod] = dynamics_mod(acc_measu, p_mean, v_mean,step ,n_samples);
%INTEGRAL(REAL)= MODEL (deterministic)

[pos_est, vel_est ] = dynamics_real(p_mean, p_var, v_mean, v_var, acc_sin,step, n_samples);
%INTEGRAL(IDEAL) = IDEAL
```

The first function refers to Equation (2), where the current acceleration from the vehicle is added with a bias and a noise.

```
function [acc_measu, bias_model, noise_model] = acc_mod(a_sin, noise_mean, noise_var, bias_mean, bias_var,n_samples)
    noise_model = normrnd(noise_mean,sqrt(noise_var),1,n_samples);      %Gaussian model, dimension 1 of each element
    bias_model = ones(1,n_samples)*normrnd(bias_mean,sqrt(bias_var));  %The bias is constant
    acc_measu = a_sin + bias_model + noise_model;                       %Real model
end
```

Note that the noise is being generated with a random sequence using the MATLAB command *'normrnd'*. Conversely, the bias is a constant random number added to all the elements of the vector. The next function computes the measured velocity and position of the vehicle using the measured acceleration from the accelerometer.

```
function [pos_mod, vel_mod] = dynamics_mod(acc_theo, po_mean, vo_mean,step,n_samples)
pos_mod= zeros(1,n_samples); %position
pos_mod(1) = po_mean;
vel_mod = zeros(1,n_samples); %velocity
vel_mod(1) = vo_mean;
    for j=2:n_samples
        vel_mod(j) = vel_mod(j-1) + acc_theo(j-1)*step;
        pos_mod(j) = pos_mod(j-1) + vel_mod(j-1)*step + 0.5*acc_theo(j-1)*(step)^2;
    end
end
```

Note that lines 7-8 represent the following kinematic equations for a Uniformly Accelerated Rectilinear Motion (UARM) using an Euler integration formula

$$
\begin{aligned}
v_c(t_{j+1}) &= v_c(t_j) + a_c(t)\Delta t \\
p_c(t_{j+1}) &= p_c(t_j) + v_c(t)\Delta t + a_c(t)\frac{\Delta t^2}{2}
\end{aligned}
\tag{17}
$$

Furthermore, a similar logic is used to obtain the true position and velocity

```
function [pos_est, vel_est] = dynamics_real(p_mean, p_var, v_mean, v_var, acc_sin,step, n_samples)
%
 p_initial = normrnd(p_mean,sqrt(p_var));
pos_est = zeros(1,n_samples);
pos_est(1) = p_initial;
%
 v_initial = normrnd(v_mean,sqrt(v_var));
 vel_est = zeros(1,n_samples);
 vel_est(1) = v_initial;

    for j=2:n_samples
        vel_est(j) = vel_est(j-1) + acc_sin(j-1)*step;
        pos_est(j) = pos_est(j-1) + vel_est(j-1)*step + 0.5*acc_sin(j-1)*((step))^2;
     end
end
```

where the Euler integration looks like this

$$
\begin{aligned}
v_E(t_{j+1}) &= v_E(t_j) + a(t)\Delta t \\
p_E(t_{j+1}) &= p_E(t_j) + v_E(t)\Delta t + a_c(t)\frac{\Delta t^2}{2}
\end{aligned}
\tag{18}
$$

with initial statistics $v(0) = v_E(0) \sim N(\overline{v}_0, M_0^v)$ and $p(0) = p_E(0) \sim N(\overline{p}_0, M_0^p)$. These results are plugged into the upcoming sections of the code for the computation of the GPS Dynamics.

4.1.3 Derivation of the Dynamic Model

The objective is to define a stochastic discrete-time system that is independent of the acceleration profile, as suggested in the assignment prompt. Therefore, the Equation 18 is subtracted from Equation 17 to obtain the following state space realization

$$
\begin{gathered}
\delta_{t_{j+1}} = \Phi\delta_{t_j} - \Gamma w(t_j) \\
\begin{bmatrix} \delta_{PE}(t_{j+1}) \\ \delta_{vE}(t_{j+1}) \\ b(t_{j+1}) \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & -\frac{\Delta t^2}{2} \\ 0 & 1 & -\Delta t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \delta_{PE}(t_j) \\ \delta_{vE}(t_j) \\ b(t_{j+1}) \end{bmatrix} - \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \\ 0 \end{bmatrix} w(t_j)
\end{gathered}
\tag{19}
$$

where the initial conditions are defined below

$$
\begin{aligned}
\delta_{PE}(t_0) &\sim N(0, M_0^p) \\
\delta_{VE}(t_0) &\sim N(0, M_0^v) \\
b &\sim N(0, M_0^b)
\end{aligned}
\tag{20}
$$

this code shows the implementation of this model

```
%% 3. Derivation of Dynamic Model
% Vector provided in the solution of the Project
deltax_r =[pos_est - pos_mod;
            vel_est - vel_mod;
```

```
5                   bias_model];
6 % For the initialization of the Kalman Filter
7 d_pe = deltax_r(1,:) ;
8 d_ve = deltax_r(2,:) ;
9 % Propagation matrixes
10 Phi = [1, step, -(1/2)*(step^2);
11         0,    1,          -step;
12         0,    0,              1];
13 Gamma = [(1/2)*(step^2);
14                    step;
15                       0];
```

where lines 3-5 show the state vector independent of the acceleration profile.

4.1.4 GPS Model

In the following lines of code, the measured position from the GPS is extracted from the current position and velocity calculated in the previous subsection.

```
1 %% 4. GPS
2 dt_gps = 1/freq_gps;
3 t_gps = 0:dt_gps:L;
4 n_samples_gps = length(t_gps);
5
6 %Equations for the GPS
7 [z_pos, z_vel, idx, eta_pos, eta_vel]=GPS_dynamics(n_samples,n_samples_gps,etap_mean,etap_var,etav_mean,etav_var,
      pos_est,vel_est);
8
9 % Difference between GPS measurement and ground truth
10 % GPS measurement that will be included in the Kalman Filter Algorithm
11 dz = [d_pe(idx) + eta_pos;
12       d_ve(idx) + eta_vel];
```

The first lines are computing a new time vector based on the sampling frequency of the GPS. The MATLAB function 'GPS_dynamics' is explained below

```
1 function [z_pos, z_vel, idx, eta_pos, eta_vel] = GPS_dynamics(n_samples, n_samples_gps, etap_mean, etap_var,
      etav_mean,etav_var, pos_est, vel_est)
2     %Index for the update of GPS measurements corresponding to
3     %Accelerometer
4     idx = 1:40:n_samples;
5     %Etas noise
6     eta_pos = normrnd(etap_mean,sqrt(etap_var),1,n_samples_gps); % This is not a bias
7     eta_vel = normrnd(etav_mean,sqrt(etav_var),1,n_samples_gps); % This is not a bias, random number
8     %gps measurements
9     z_pos = pos_est(idx) + eta_pos;
10    z_vel = vel_est(idx) + eta_vel;
11 end
```

A real GPS is a very accurate device, yet it has to include in its modeling the effects of a Gaussian noise for each of the true velocity a position. The output dz on line 9 will be identified as the "measured" variables from the GPS.

4.1.5 Kalman Filter: Initialization

Prior to starting with the updating process for each sample time, it is important to initialize each of the variables and matrices in Equations (8-11) explained in subsection (3.4).

```
%% 5. Kalman Filter Algorithm
%% 5.A Initialization
% Prior Knowledge
I_dimKal = Phi^(0);
```

```
W = noise_var;
% Equations taken from Chp4 — Slide 39
% GPS Covariance
V = [etap_var          0  ;
          0    etav_var  ];
% C/H matrix Output
H = [1 0 0;
     0 1  0]; % We are mutting the Bias Channel
% Update
% Initial  Values
% (known) Covariance
M0 = [p_var      0             0;
          0 v_var             0;
          0        0  bias_var ];
% Slide 43
x0_mean =      [  d_pe(1);
                  d_ve(1);
         bias_model(1)];
%
K0 = M0*H'*(H*M0*H'+V)^(-1);
%(known) A posteriori Covariance, Slide 48
P0 = (I_dimKal-K0*H)*M0*(I_dimKal-K0*H)' +  K0*V*K0';
% Estimate state, Slide  49
x0_est  = x0_mean + K0*(dz(:,1)-H*x0_mean);
%
 x_est_total   = zeros(3,n_samples);
 x_est_total (:,1)  = x0_est;
x_mean_total = zeros(3,n_samples);
x_mean_total(:,1)  = x0_mean;
% Matrices
M_total{1,1} = M0;
K_total{1,1} = K0;
P_total{1,1} = P0;
```

Note that the matrix H is defined in such a way that the channel dedicated to the bias is muted because is not being measured. Moreover, there is no covariance among the position, velocity, and bias in matrices M_k, P_k, P_k(off-diagonal elements equal to zero).

4.1.6 Kalman Filter: Real-Time Algorithm

The code below basically shows the backbone of the Kalman Filter algorithm. It is running from 2 sampling instants to the number of samples that correspond to the frequency of the accelerometer ($T = 30$) because the first instant in for the initial conditions. Lines 17 to 24 explicitly show the updated algorithm considered in Slide 43-Chapter 4 of our lecture notes. Note that the 'if' condition in line 9 is playing a critical role in the proper calculation of the gain and matrices. The values are being updated every 40 sampling instants because of the ratio between the GPS frequency (5 Hz) and the accelerometer one (200 Hz).

```
1  %% 5.B Real-Time Algorithm
2  for  i=2:n_samples
3      M_prev = M_total{1,i-1};
4      K_prev = K_total{1,i-1};
5      P_prev = P_total{1,i-1};
6      % Equation of Slide 43 — Propagation
7      x_mean_total(:,i)  = Phi*x_est_total (:, i-1);
8      % Verification  if  this  matches the sampling time of the Accelerometer
9          if    mod(i-1, 40) ~= 0
10             %Fixing the matrixes if that  doesn't  match
11             x_est_total (:, i)  = x_mean_total(:,i);
12            M_total{1,i} = M_prev;
```

```
            K_total{1,i} = K_prev;
            P_total{1,i} = P_prev;
        else
           %Slide 43
            M_next = Phi*P_prev*Phi'+Gamma*W*Gamma';
            K_next = M_next*H'*(H*M_next*H'+V)^(-1);
            P_next = (I_dimKal - K_next*H)*M_next*(I_dimKal - K_next*H)'+ K_next*V*K_next';
            % Update
            M_total{1,i} = M_next;
            K_total{1,i} = K_next;
            P_total{1,i} = P_next;
            x_est_total (:, i) = x_mean_total(:,i) + K_next*(dz(:,(i-1)/40+1) - H*x_mean_total(:,i));
        end
    end
```

4.1.7 Data collection

These are the output vectors obtained from the Kalman Filter Algorithm. Each element of each vector is extracted to provide separated plots for the velocity, position, and bias

```
% Final vectors
% Estimated Position Mean
final_p_est  = x_est_total (1,:) ;
final_v_est  = x_est_total (2,:) ;
final_b_est  = x_est_total (3,:) ;

% Position Mean
final_p_prior  = x_mean_total(1,:);
final_v_prior  = x_mean_total(2,:);
final_b_prior  = x_mean_total(3,:);

for i=1:n_samples
    P_current = P_total{1,i};
    pos_est_var_cell {1,i} = P_current(1,1);
    vel_est_var_cell {1,i} = P_current(2,2);
    bias_est_var_cell {1,i}= P_current(3,3);
end

%Variance of the Estimated Position
pos_est_var  = cell2mat( pos_est_var_cell );
vel_est_var  = cell2mat( vel_est_var_cell );
bias_est_var  = cell2mat( bias_est_var_cell );

% Prior error
e_priori  = deltax_r - x_mean_total;
e_posteriori  = deltax_r - x_est_total;
```

4.1.8 Monte Carlo Approach: Ensemble of realizations

After completing the analysis for one realization, the process is repeated for different epochs, so that the filter performance improves as the prior knowledge after each iteration is updated. Moreover, with these ensembled realizations the error variance, the orthogonality property among the errors, and the residuals are computed and graphed. The epochs are selected to increase 4 times each other in this way: [10 40 160 640]. The code for this experiment is provided in Appendix 2.

After initializing all the parameters from the previous code, we can initialize the global vectors that are going to store all the results after each epoch.

```
%% Epochs
Epochs = [10 40 160 640];
```

```matlab
3 Final_Table_Properties=zeros(length(Epochs),9);
4 time = zeros(1,length(Epochs));
5 for it = 1:length(Epochs)
6     tic;
7     dim = Epochs(it);
8     e_posteri_global = zeros(3,n_samples,dim);
9     x_mean_global = zeros(3,n_samples,dim);
10    x_est_global = zeros(3,n_samples,dim);
11    dz_total = zeros(2,n_samples_gps,dim);
12    orth_res = zeros(2,2,dim);
13    for j=1:dim
14        [acc_measu, bias_model, noise_model] = acc_mod(acc_sin, noise_mean, noise_var, bias_mean, bias_var,n_samples); %REAL + BIAS + NOISE = MEASURED
15        [pos_mod, vel_mod] = dynamics_mod(acc_measu, p_mean, v_mean,step ,n_samples);             %INTEGRAL(MEASURED) = Model, there is no variances here (deterministic)
16        [pos_est, vel_est] = dynamics_real(p_mean, p_var, v_mean, v_var, acc_sin,step, n_samples);  %INTEGRAL(REAL) = Estimate (stochastic)
17        % Delta of state variables
18        dx_real =[pos_est - pos_mod;
19                  vel_est - vel_mod;
20                        bias_model];
21        d_pe = dx_real(1,:);
22        d_ve = dx_real(2,:);
23        % GPS
24        [z_pos, z_vel, idx, eta_pos, eta_vel] = GPS_dynamics(n_samples, n_samples_gps, etap_mean, etap_var, etav_mean,etav_var, pos_est, vel_est);
25        % Difference between GPS measurement and ground truth
26        dz = [d_pe(idx) + eta_pos;
27              d_ve(idx) + eta_vel];
28        dz_total(:,:, j)=dz; %%%%%%%%%%
29        % Initialization
30        %
31        x0_mean =     [ d_pe(1);
32                        d_ve(1);
33                  bias_model(1)];
34        x0_est = x0_mean + K0*(dz(:,1)-H*x0_mean);
35        %
36        x_est_total = zeros(3,n_samples);
37        x_est_total(:,1) = x0_est;
38        x_mean_total = zeros(3,n_samples);
39        x_mean_total(:,1) = x0_mean;
40        % Matrices
41        M_total{1,1} = M0;
42        K_total{1,1} = K0;
43        P_total{1,1} = P0;
44        %%%%%%%%%%%%%%%%%%%%%%%%%%%
```

From lines 6-10 these global vectors are initialized according to the current epoch selected. Note that on line 5 we use the 'tic' command of MATLAB to take the runtime. After doing the real-time Kalman algorithm, we store the values in the same way as before but we are including the results of each epoch in a global vector (lines 27-29)

```matlab
1 time(it)=toc;
2 % Final vectors
3 final_p_est = x_est_total(1,:);
4 final_v_est = x_est_total(2,:);
5 final_b_est = x_est_total(3,:);
6
7 final_p_prior = x_mean_total(1,:);
8 final_v_prior = x_mean_total(2,:);
9 final_b_prior = x_mean_total(3,:);
10
```

```
11 for k=1:n_samples
12     P_current = P_total{1,k};
13      pos_est_var_cell {1,k} = P_current(1,1);
14      vel_est_var_cell {1,k} = P_current(2,2);
15      bias_est_var_cell {1,k}= P_current(3,3);
16 end
17
18 pos_est_var  = cell2mat( pos_est_var_cell );
19  vel_est_var  = cell2mat( vel_est_var_cell );
20  bias_est_var  = cell2mat( bias_est_var_cell );
21
22 % Priori error
23  e_posteriori  = dx_real − x_est_total;
24
25 % Total matrices
26  e_posteri_global (:,:, j)= e_posteriori;% errors
27  x_est_global (:,:, j)= x_est_total;
28 x_mean_global (:,:, j)=x_mean_total;
29 end
```

Moreover, after a epoch is finalized all the errors are averaged and the maximum value is stored in the matrices from lines 10-12

```
1 %  1.Property: Average of the ensemble error
2 e_pave = sum(e_posteri_global(1,idx,:) ,  3)/dim;
3 e_vave = sum(e_posteri_global(2,idx,:) ,  3)/dim;
4 e_bave = sum(e_posteri_global(3,idx,:) ,  3)/dim;
5
6 e_ave = [e_pave;
7          e_vave;
8         e_bave];
9
10 Prop1p = max(max(e_pave));
11 Prop1v = max(max(e_vave));
12 Prop1b = max(max(e_bave));
```

Next, the variance of the estimated error is compared with the error that the filter obtains for each sampling instant, for each realization. This is the mathematical expression for the implemented code

$$\overline{\sigma^2} = \frac{1}{N-1}\sum_{N}^{k=1}[e^k(t_i) - \overline{e}(t_i)][e^k(t_i) - \overline{e}(t_i)]^T \qquad (21)$$
$$\overline{\sigma^2} - \sigma^2 \approx 0$$

for any instant $i > 0$, and $k \leq N$, the number of ensemble of realizations. In line 14, the a posteriori error is subtracted from the average error computed in the previous section. Then in line 23, the result is averaged over the number of epochs, as it is done in Equation (21). Finally, this result is subtracted from the matrix P_k that has all the variances from the system.

Furthermore, the orthogonal property

$$\frac{1}{N}\sum_{N}^{k=1}[e^k(t_i) - \overline{e}(t_i)][< \tilde{\delta_x}]^T \approx 0 \qquad (22)$$

and for the residuals

$$r^l(t_i) = \frac{1}{N}\sum_{N}^{k=1} r^l(t_i) r^l(t_m) \approx 0, \, \forall i < m \qquad (23)$$

are implemented in lines 18 and lines 35-40, respectively (lines 13-21)

```
1  % 2.Property: (Error Variance vs Filter) + Orthogonality Properties
2   delta = zeros(3,n_samples_gps,dim);
3   P_var_av = zeros(3,3,n_samples_gps);
4
5   err_var = P_var_av;
6   Diff_var = P_var_av;
7
8   orth = P_var_av;
9   Orth_var = P_var_av;
10
11  for x = 1:n_samples_gps
12      for y = 1:dim
13      %2.Property: Second Order Stadistics
14      delta (:, x,y) = e_posteri_global (:, x,y)- e_ave(:,x);
15      err_var (:,:, y) = delta(:,x,y)*delta (:, x,y) '; %Ortogonalidad de los errores
16
17      %3.Property: Independece + orthogonality E[()x']
18      orth (:,:, y) = delta(:,x,y)*x_est_global (:, x,y) ';
19      end
20
21      %2. Second Order Stadistics
22      P_var_av (:,:, x)    = sum(err_var,3)/(dim-1); %Average del Error Variance
23      Diff_var (:,:, x)    = P_var_av (:,:, x) - P_total{1,x}; % Differencia entre el Error Variance y el que te bota el
    filtro , Esto tiene que ser zero
24
25      %3
26      Orth_var (:,:, x) = sum(orth,3)/(dim-1);
27  end
28
29  %
30  Prop2 = max(max(max(Diff_var)));
31  Prop3= max(max(max(Orth_var)));
32
33  % 4.Property: Residuals
34
35  for f=1:dim
36      res = dz_total (:,:, f) - x_mean_global(1:2, idx, f); % medicion del GPS -
37      orth_res (:,:, f) = res(:,13+1)*res(:,13) '; % verificar entre un tiempo y el tiempo siguiente, esto tiene que ser
    zero
38  end
39  Res_ave (:,:) = (1/dim)*sum(orth_res,3);
40  Prop4_res = max(max(max(Res_ave)));
```

We are considering a sample in instant 13 to validate the independence of its residuals from the previous ones (line 37).

4.2 Results

The values obtained by the use of the previous code are plotted and also zoomed in to obtain a better perspective of what is happening from each sampling time to the next one

- **Plot for the Accelerometer's dynamics:**

  Figure (2) presents the difference between the current state of the variables (from the Euler integration) and the measured by the accelerometer. Notice from the zoom-in figures on each plot that the measured velocity and position are very close to the calculated one by the Euler integration of acceleration profile. It will be shown later that this offset will increase as time goes on. Furthermore, the noise and bias are explicitly affecting each measurement of the acceleration, resulting in a constant offset along the experiment length. Therefore, in the third plot zoom in, we can see the measurements of a real accelerometer.

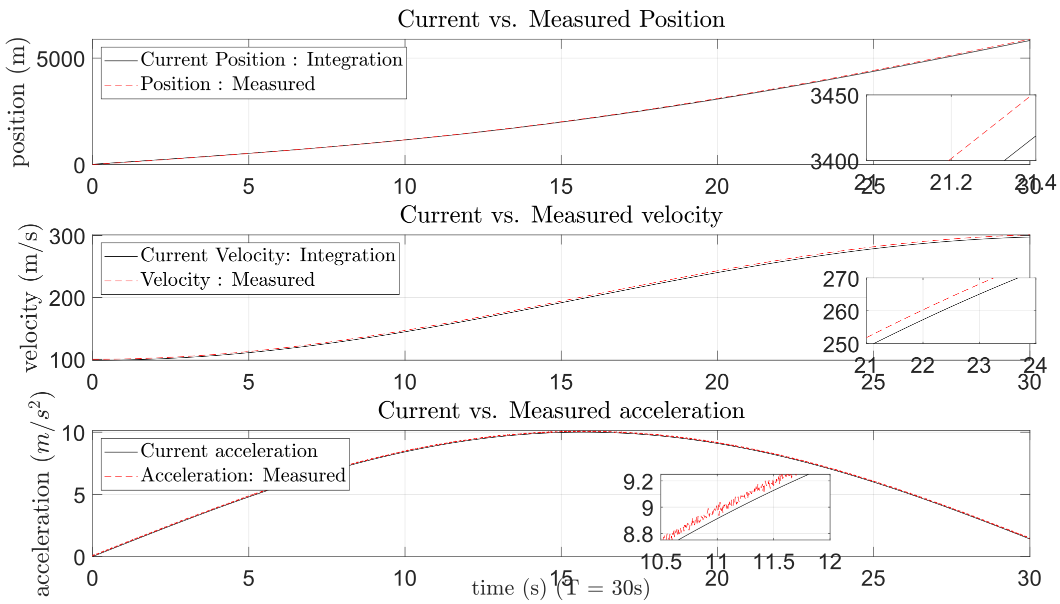

Figure 2: Accelerometer's Dynamics

- **Accelerometer's dynamics**

  Figure (3) presents on the left the comparison of plots for the velocity and position between the GPS measurements (affected by η_0) and the calculated ones from the Euler integration. From a first impression of the left plots and their respective zoom-in plots, these plots match. However, if we compute the difference between these two plots, we would be capable of reproducing the η 's in Equation (3) - Plots to the right.

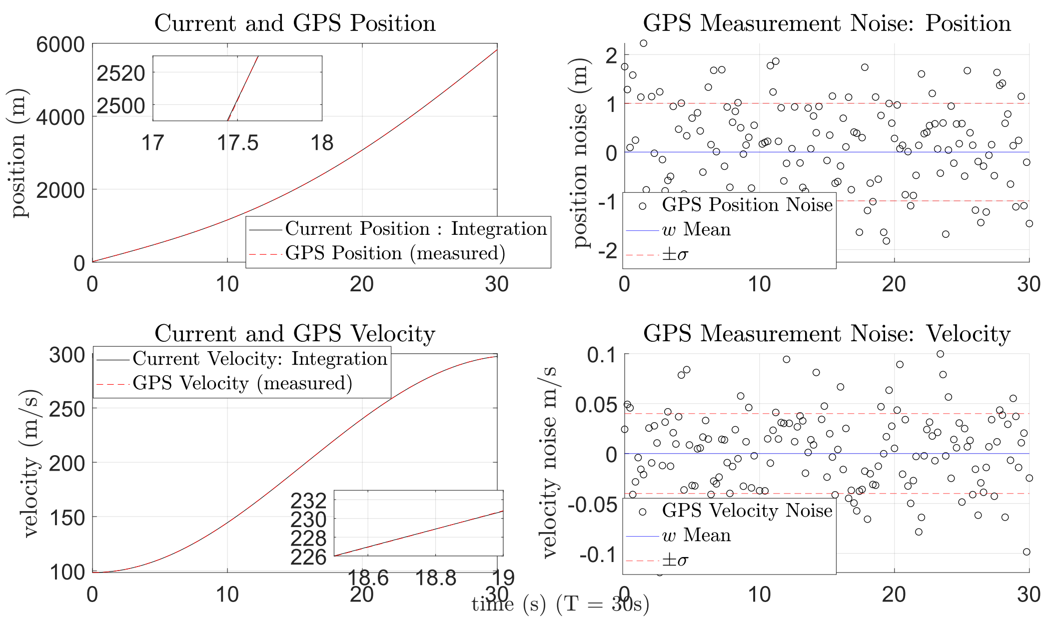

Figure 3: GPS' dynamics

- **Plot for the difference between the three models (GPS vs.Euler vs.Measured)**

  Figure (4) presents the pairwise comparison of each approach. We can draw a straightforward interpretation from the results that are in the yellow plot: the difference between the current position and velocity and the measured by the GPS oscillates around 0 because of Equation(3).

  For the blue plots, these patterns occur because the measured velocity and position integrate the bias and error of the measured acceleration. For the velocity, integrating once a constant causes a linear pattern. For the position, integrating twice a constant produces a quadratic pattern.

  Finally, the red dashed plot is equal to the previous one but with an additive noise.

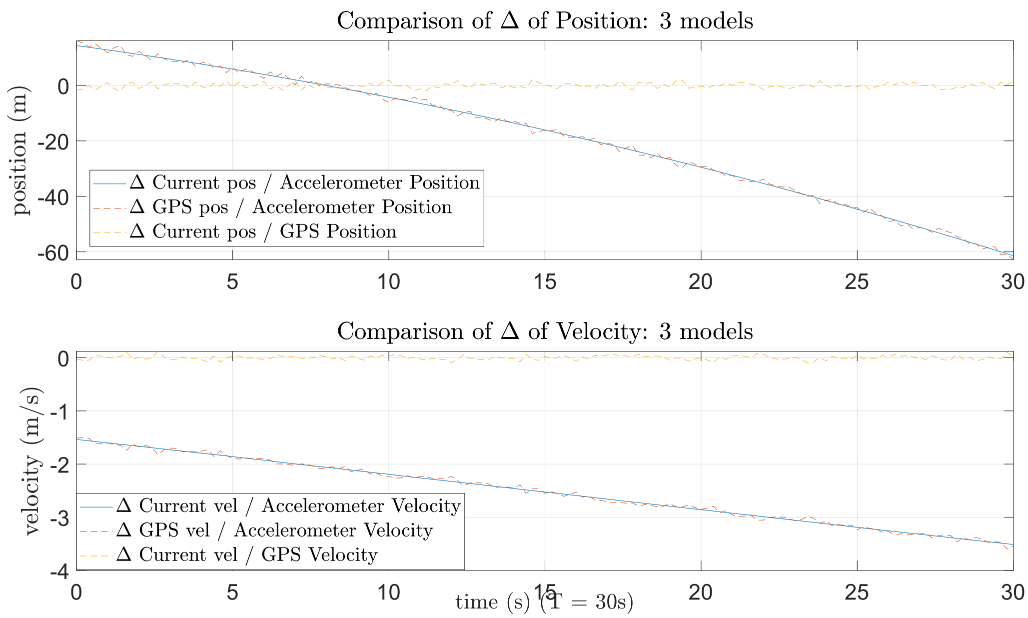

Figure 4: Comparision of the GPS, Accelerometer, and Euler models for the system

- **Kalman Filter Results**

  The following plots show the successful implementation of a I-D Kalman Filter. Figure (5) shows how the estimated delta of the position, velocity, and bias is able to 'follow' their errors of their corresponding variables. Moreover, the sigma bound lies within reasonable proximity to the blue plots. It is worth mentioning that the variance for the bias in the third plot in Figure (5) decreases reasonably fast in the experiment length. Almost in t=28s, the estimated bias reached the real bias. Figure (6) validates these arguments by showing that the variance ('uncertainty') decreases. The third plot in this figure shows that the bias takes more time in decreasing to zero. Figure (7) shows that all the errors are reduced considerably to zero.

- **Apriori vs. Aposteriori**

  Figure (8) presents the relationship between the a priori and a posteriori results. From the equation (13), we can affirm that one plot is delayed with respect to the other because the update process occurs just every 40 sampling instants. Moreover, the term $K_k r_k$ causes a small difference between the $\overline{x}$ term and $\tilde{x}$

- **Emsemble of realizations**

  Figure (9) presents the average error for each number of epochs selected, for each sample time. It can be clearly seen that the system improves its estimation because the a priori statistics are updated

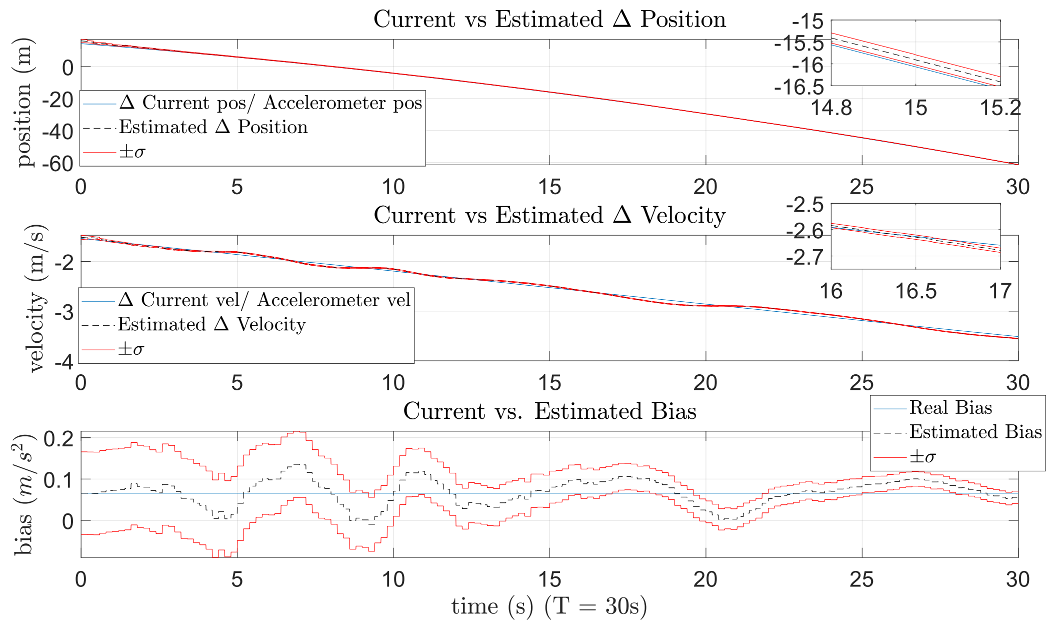

Figure 5: Comparison of the Δ of the current and estimated variables of the system

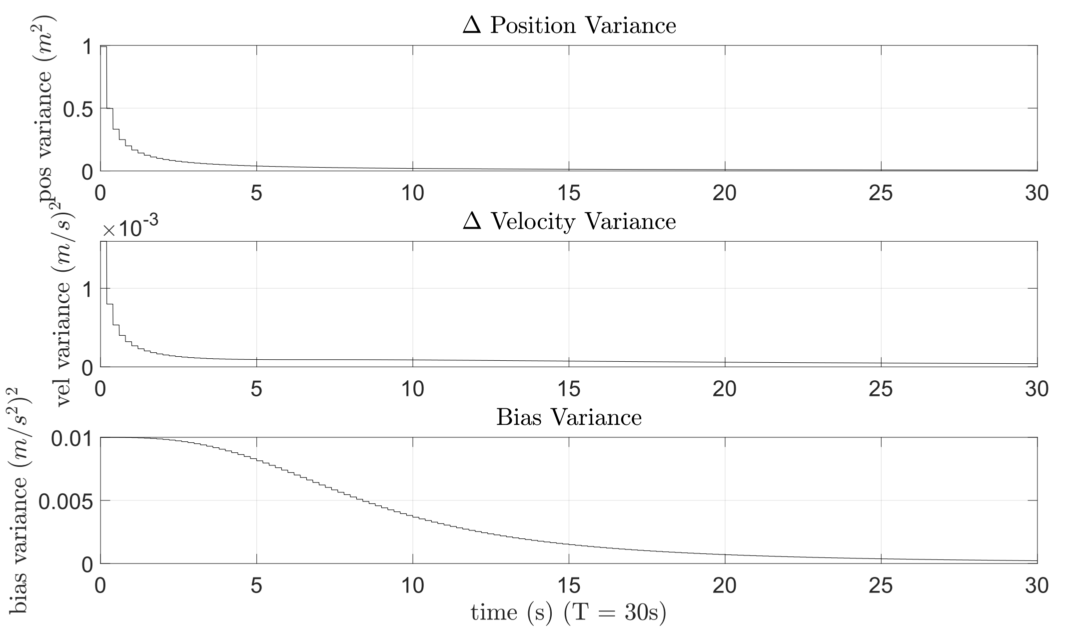

Figure 6: Evolution of the Δ variances

and the error for each of the variables of interest reduces to zero. The position takes more time to settle than the velocity and the bias. Moreover, the computation for each ensemble realization experiment is included in each legend. The algorithm improves reasonably well for a short number of epochs and increases the calculation linearly. After 640 epochs the error is nearly 0 for the average position error (purple line).

- **Error variance and Orthogonality properties**

The first plot of Figure (10) presents the variance for the comparison between the estimation error

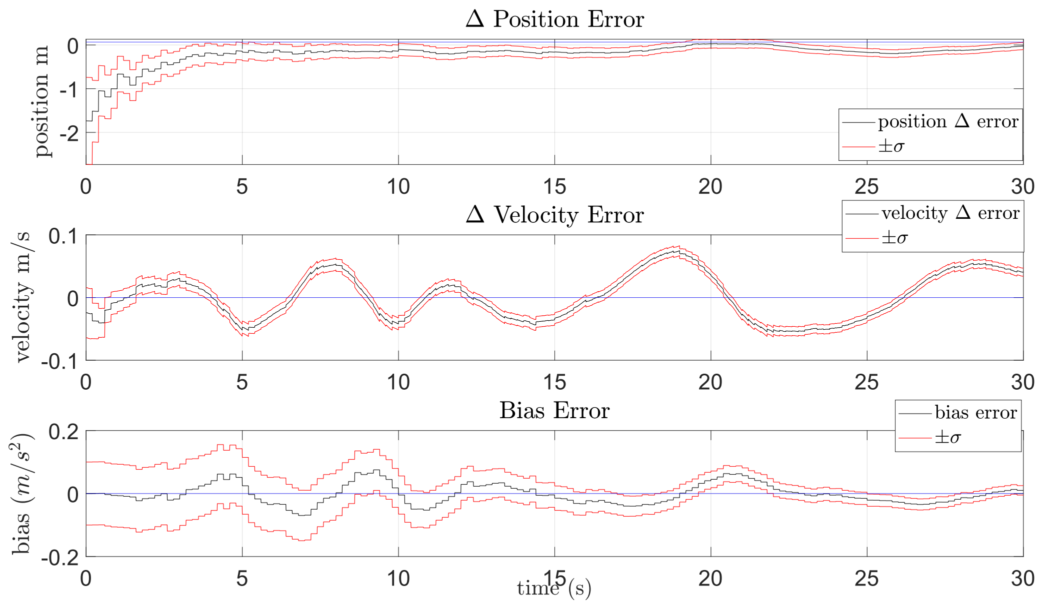

Figure 7: Δ of the errors for each of the variables of the system

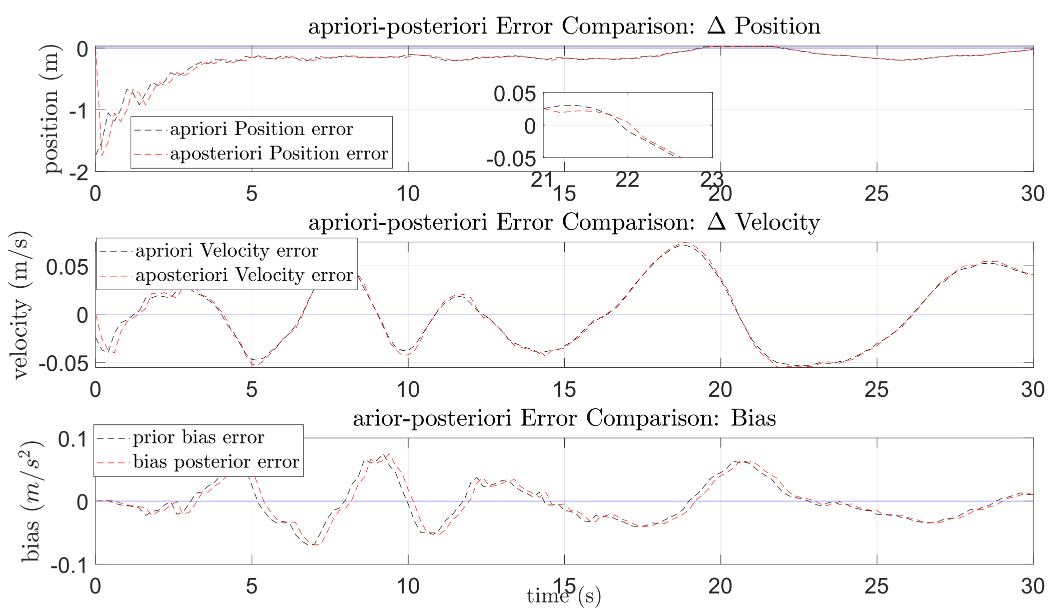

Figure 8: A priori - a posteriori errors

and the mean error, also introduced in Equation (21). This result proves that our Kalman filter is properly implemented because the simulated error variance is close to the filter error variance. Moreover, the results of the Equation (22) and (23) for the orthogonality properties are presented in the second and third plots of Figure (10). Since the experimental results are near zero, this validates the theoretical independence of the errors and the residuals, demonstrating that this is a Markov process. Notice how as the number of epochs increases, the error is reduced considerably. It was foreseeable that for 10 epochs the errors were the highest from the batch because the filter doesn't

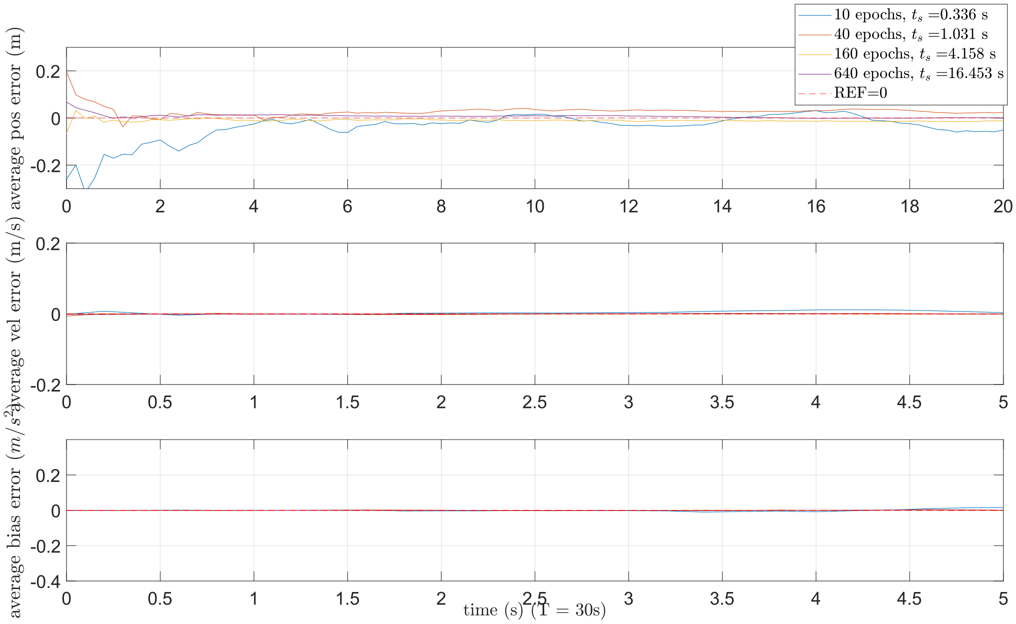

Figure 9: Results of the Errors after the ensembled realization experiments

have enough a priori information to reduce the error.

Figure 10: Error variances and orthogonality properties

5 Conclusions

This manuscript shows the step-by-step implementation of a Kalman filter for the estimation of the position and velocity of a car. For this purpose, an accelerometer is used to measure the acceleration, and then, through an Euler integration, the position and velocity are derived. Since this electronic device is affected by white noise and bias, our job is to calibrate it using a GPS. It was shown in Figure (5) that the error increases between the current state of the car and the accelerometer's measurements as time goes on. Therefore, a linear dynamic system is derived, independent from the acceleration - the model is presented in Equations (19) and (20). This work has attained to correctly model the propagation of the system dynamics and the update of the estimations through the GPS measurements for one realization and for ensembled realizations of [10, 40, 160, 160] epochs. The objectives at the beginning of this work are revised:

1. *Determine an associated stochastic discrete-time system that is approximately independent of the acceleration profile.*

   (Result). The system that is independent of the acceleration is presented in equations 19 and 20 and then Implemented in section (4.1.3).

2. *Design a Kalman filter and plot the estimates of position, velocity, an accelerometer bias as well as the filter error variance (considering one sigma bound)*

   (Result). It was shown in the plots of Figure (5)-(6)-(7) how the Kalmar filter is capable of correctly estimating the delta od the position, velocity and bias with very low uncertainty (1 σ bound), and to reduce its error as time goes (7).

3. *For the validation of the minimum variance estimator, we should follow a Monte Carlo approach to do ensemble realizations. Specifically, the batches proposed are* [10 40 160 640] *realizations*

   (Result). It was shown in the plots of Figure (9) how the filter is capable of reducing its average error over the experiment length for ensembled realizations. Indeed, the average error for 640 epochs is almost 0.

   - *Proof that over an ensemble of realizations, the simulated error variance is close to the filter error variance*

     (Result). It was shown in the top plot of Figure (10) how the theoretical error variance is close to the variance that the Kalman filter outputs. Furthermore, as the epoch increases, this error is almost reduced to zero.

   - *Show the theoretical orthogonalities and independence properties are satisfied for this filter*

     (Result). It was shown in the last two plots of Figure (10) how the orthogonal properties behave for the estimated error and the residuals. These errors were close to zero for a sampling instant '13' and they reduced when the epoch increases.

Even though this project was implemented for a 1-D case, it will help the reader to extend the scope to a 3-D case. Thus, further work could extend the implementation of this 1-D Kalman filter to a 3-D Kalman filter for more realistic applications for the industry.

6 Appendix

6.1 Code to implement one realization of the Kalman Filter

```
%% 1. Variable Initialization
clc
clear all
% 1.A Accelerometer / Data from the problem
s_rate = 200;        % 1/s,    Variable 1
step =(1/s_rate);
Ampl = 10;           % m/s^2, Variable 2
L = 30 ;             % s, Experiment length,Variable 3
w = 0.1  ;           % f, 2pi,Variable 4

% 1.B Accelerometer Truth
t_sin = [0:(step):L];
acc_sin = Ampl*sin(w*t_sin);
n_samples = length(t_sin);

% 1.C Accelerometer Measured: Bias + Noise
bias_mean = 0;       %(m/s^2)^(2),Variable 5
noise_mean = 0;      %(m/s^2)^(2),Variable 6

bias_var = 0.01;     %(m/s^2)^(2),Variable 7
noise_var = 0.0004; %(m/s^2)^(2),Variable 8

% 1.D GPS, 'Current' Dynamics: Position & Velocity
% prior position
freq_gps = 5;        %Hz, synchronized with accelerometer,Variable 9
p_mean = 0;          % 0 meters,Variable 10
p_var = 100;         % (10 meters)^2,Variable 11
% prior velocity
v_mean = 100;        % 10 meters/s,Variable 14
v_var = 1;           % (1 meters/s)^(2),Variable 15

%A priori stadistics for the dynamic noise processes
etap_mean = 0;         %m,Variable 12
etap_var = 1;          %m^2,Variable 13
%velocity noise
etav_mean = 0;         %cm/m,Variable 16
etav_var = (0.04)^2;   %convert(4cm/s)^2 to (m/s)^2,,Variable 17

%% 2. Accelerometer
[acc_measu, bias_model, noise_model] = acc_mod(acc_sin, noise_mean, noise_var, bias_mean, bias_var,n_samples); %IDEAL
    +BIAS+NOISE=REAL
[pos_mod, vel_mod] = dynamics_mod(acc_measu, p_mean, v_mean,step ,n_samples);%INTEGRAL(REAL)= MODEL (
    deterministic)

[pos_est, vel_est ] = dynamics_real(p_mean, p_var, v_mean, v_var, acc_sin,step, n_samples);  %INTEGRAL(IDEAL) =
    IDEAL

%% 3. Derivation of Dynamic Model
% Vector provided in the solution of the Project
deltax_r =[pos_est - pos_mod;
           vel_est - vel_mod;
                 bias_model];
% For the initialization of the Kalman Filter
d_pe = deltax_r(1,:) ;
d_ve = deltax_r(2,:) ;
% Propagation matrixes
Phi = [1, step, -(1/2)*(step^2);
```

```
        0,      1,                  -step;
        0,      0,                      1];
Gamma = [(1/2)*(step^2);
                          step;
                             0];
%% 4. GPS
dt_gps = 1/freq_gps;
t_gps  = 0:dt_gps:L;
n_samples_gps = length(t_gps);

%Equations for the GPS
[z_pos, z_vel, idx, eta_pos, eta_vel] = GPS_dynamics(n_samples, n_samples_gps, etap_mean, etap_var, etav_mean,
      etav_var, pos_est, vel_est);

% Difference between GPS measurement and ground truth
% GPS measurement that will be included in the Kalman Filter Algorithm
dz = [d_pe(idx) + eta_pos;
        d_ve(idx) + eta_vel];
%% 5. Kalman Filter Algorithm
%% 5.A Initialization
% Prior Knowledge
I_dimKal = Phi^(0);
W = noise_var;
% Equations taken from Chp4 - Slide 39
% GPS Covariance
V = [etap_var          0  ;
            0    etav_var  ];
% C/H matrix Output
H = [1 0 0;
      0 1  0]; % We are mutting the Bias Channel
% Update
% Initial  Values
% (known) Covariance
M0 = [p_var      0             0;
               0 v_var             0;
               0        0  bias_var];
% Slide 43
x0_mean =      [ d_pe(1);
                   d_ve(1);
             bias_model(1)];
%
K0 = M0*H'*(H*M0*H'+V)^(-1);
%(known) A posteriori Covariance, Slide 48
P0 = (I_dimKal-K0*H)*M0*(I_dimKal-K0*H)' +  K0*V*K0';
% Estimate state, Slide 49
x0_est  = x0_mean + K0*(dz(:,1)-H*x0_mean);
%
 x_est_total  = zeros(3,n_samples);
 x_est_total (:,1)  = x0_est;
x_mean_total = zeros(3,n_samples);
x_mean_total(:,1)  = x0_mean;
% Matrices
M_total{1,1} = M0;
K_total{1,1} = K0;
P_total{1,1} = P0;
%% 5.B Real-Time Algorithm
for  i=2:n_samples
     M_prev = M_total{1,i-1};
     K_prev = K_total{1,i-1};
     P_prev = P_total{1,i-1};
     % Equation of Slide 43 - Propagation
```

```
    x_mean_total(:,i) = Phi*x_est_total(:, i-1);
    % Verification if this matches the sampling time of the Accelerometer
        if  mod(i-1, 40) ~= 0
             %Fixing the matrixes if that doesn't match
            x_est_total (:, i) = x_mean_total(:,i);
            M_total{1,i} = M_prev;
            K_total{1,i} = K_prev;
            P_total{1,i} = P_prev;
        else
           %Slide 43
            M_next = Phi*P_prev*Phi'+Gamma*W*Gamma';
            K_next = M_next*H'*(H*M_next*H'+V)^(-1);
            P_next = (I_dimKal - K_next*H)*M_next*(I_dimKal - K_next*H)'+ K_next*V*K_next';
            % Update
            M_total{1,i} = M_next;
            K_total{1,i} = K_next;
            P_total{1,i} = P_next;
            x_est_total (:, i) = x_mean_total(:,i) + K_next*(dz(:,(i-1)/40+1) - H*x_mean_total(:,i));
        end
    end

% Final vectors
final_p_est  = x_est_total (1,:) ;
final_v_est  = x_est_total (2,:) ;
final_b_est  = x_est_total (3,:) ;

final_p_prior  = x_mean_total(1,:);
final_v_prior  = x_mean_total(2,:);
final_b_prior  = x_mean_total(3,:);

for i=1:n_samples
    P_current = P_total{1,i};
     pos_est_var_cell {1,i} = P_current(1,1);
     vel_est_var_cell {1,i} = P_current(2,2);
     bias_est_var_cell {1,i}= P_current(3,3);
end

pos_est_var  = cell2mat( pos_est_var_cell );
vel_est_var  = cell2mat( vel_est_var_cell );
bias_est_var  = cell2mat( bias_est_var_cell );

% Prior error
e_priori  = deltax_r - x_mean_total;
e_posteriori  = deltax_r - x_est_total;

%% Functions

%[pos_est, vel_est , po, vo]
function [pos_est, vel_est ] = dynamics_real(p_mean, p_var, v_mean, v_var, acc_sin,step, n_samples)
%
p_initial  = normrnd(p_mean,sqrt(p_var));
pos_est = zeros(1,n_samples);
pos_est(1) = p_initial;
%
v_initial  = normrnd(v_mean,sqrt(v_var));
vel_est = zeros(1,n_samples);
vel_est (1) = v_initial;

    for j=2:n_samples
        vel_est (j) = vel_est(j-1) + acc_sin(j-1)*step;
        pos_est(j) = pos_est(j-1) + vel_est(j-1)*step + 0.5*acc_sin(j-1)*((step))^2;
```

```
    end
end

function [pos_mod, vel_mod] = dynamics_mod(acc_theo, po_mean, vo_mean,step,n_samples)
pos_mod= zeros(1,n_samples); %position
pos_mod(1) = po_mean;
vel_mod = zeros(1,n_samples); %velocity
vel_mod(1) = vo_mean;
    for j=2:n_samples
        vel_mod(j) = vel_mod(j-1) + acc_theo(j-1)*step;
        pos_mod(j) = pos_mod(j-1) + vel_mod(j-1)*step + 0.5*acc_theo(j-1)*(step)^2;
    end
end

function [acc_measu, bias_model, noise_model] = acc_mod(a_sin, noise_mean, noise_var, bias_mean, bias_var,n_samples)
    noise_model = normrnd(noise_mean,sqrt(noise_var),1,n_samples); %Gaussian model, dimension 1 of each element
    bias_model = ones(1,n_samples)*normrnd(bias_mean,sqrt(bias_var));    %The bias is constant
    acc_measu = a_sin + bias_model + noise_model;                        %True model
end

% Is going to measure corrupted measures of the velocity
function [z_pos, z_vel, idx, eta_pos, eta_vel] = GPS_dynamics(n_samples, n_samples_gps, etap_mean, etap_var,
     etav_mean,etav_var, pos_est, vel_est)
    %Index for the update of GPS measurements corresponding to
    %Accelerometer
    idx = 1:40:n_samples;
    %Etas noise
    eta_pos = normrnd(etap_mean,sqrt(etap_var),1,n_samples_gps); % This is not a bias
    eta_vel = normrnd(etav_mean,sqrt(etav_var),1,n_samples_gps); % This is not a bias, random number
    %gps measurements
    z_pos = pos_est(idx) + eta_pos;
    z_vel = vel_est(idx) + eta_vel;
end
```

6.2 Code to implement ensembled realizations for the Kalman Filter

```
%% 1. Variable Initialization
clc
clear all
close all
font_S = 16;
% 1.A Accelerometer / Data from the problem
s_rate = 200;     % 1/s
step =(1/s_rate);
Ampl = 10;        % m/s^2
L = 30 ;          % s, Experiment length
w = 0.1  ;        % f, 2pi
% 1.B Acceleration Function
t_sin = [0:(step):L];
acc_sin = Ampl*sin(w*t_sin);
n_samples = length(t_sin);
% 1.C Accelerometer Truth: Bias + Noise
bias_mean = 0;      %m/s^2
noise_mean = 0;         %m/s^2

bias_var = 0.01;        %(m/s^2)^(2)
noise_var = 0.0004;     %(m/s^2)^(2)
% 1.D GPS, 'True' Dynamics: Position & Velocity
% prior position
p_mean = 0;         % 0 meters
p_var = 100;        % (10 meters)^2
% prior velocity
```

```
v_mean = 100;        % 10 meters/s
v_var = 1;           %   (1 meters/s)^(2)

%a priori  stadistics
etap_mean = 0;        %m
etap_var = 1;         %m^2
%velocity noise
etav_mean = 0;        %cm/m
etav_var = (0.04)^2; %convert(4cm/s)^2  to (m/s)^2

freq_gps  = 5; %Hz, synchronized with accelerometer
dt_gps = 1/freq_gps;
t_gps  = 0:dt_gps:L;
n_samples_gps = length(t_gps);
% Propagation matrixes
Phi = [1, step, -(1/2)*(step^2);
       0,    1,           -step;
       0,    0,               1];
Gamma = -1*[(1/2)*(step^2); %%%%%%%%%%%%%%%%%%%%%
                 step;
                    0];
I_dimKal =  Phi^(0);
% Prior Knowledge
W = noise_var;
% Equations taken from Chp4 - Slide 39
% GPS Covariance
V = [etap_var         0  ;
         0    etav_var  ];
% C/H matrix Output
H = [1 0 0;
     0 1  0];  % We are mutting the Bias Channel
% Update
% Initial  Values
% (known) Covariance
M0 = [p_var      0            0;
          0  v_var            0;
          0        0  bias_var ];
%
K0 = M0*H'*(H*M0*H'+V)^(-1);
%(known) A posteriori Covariance, Slide 48
P0 = (I_dimKal-K0*H)*M0*(I_dimKal-K0*H)' +  K0*V*K0';

%% Epochs
Epochs = [10 40 160 640];
Final_Table_Properties=zeros(length(Epochs),9);
time = zeros(1,length(Epochs));
for  it  = 1:length(Epochs)
    tic ;
    dim = Epochs(it);
     e_posteri_global  = zeros(3,n_samples,dim);
    x_mean_global = zeros(3,n_samples,dim);
     x_est_global  = zeros(3,n_samples,dim);
     dz_total  = zeros(2,n_samples_gps,dim);
     orth_res  = zeros(2,2,dim);
     for  j=1:dim
         [acc_measu, bias_model, noise_model] = acc_mod(acc_sin, noise_mean, noise_var,  bias_mean, bias_var ,n_samples); %REAL + BIAS + NOISE = MEASURED
         [pos_mod, vel_mod] = dynamics_mod(acc_measu, p_mean, v_mean,step ,n_samples);            %INTEGRAL(MEASURED) = Model, there is no variances here (deterministic)
         [pos_est ,  vel_est ]  = dynamics_real(p_mean, p_var, v_mean, v_var, acc_sin,step,  n_samples);   %INTEGRAL(REAL) = Estimate (stochastic)
```

```
        % Delta of state variables
        dx_real =[pos_est - pos_mod;
                  vel_est - vel_mod;
                        bias_model];
        d_pe = dx_real(1,:);
        d_ve = dx_real(2,:);
        % GPS
        [z_pos, z_vel, idx, eta_pos, eta_vel] = GPS_dynamics(n_samples, n_samples_gps, etap_mean, etap_var,
    etav_mean,etav_var, pos_est, vel_est);
        % Difference between GPS measurement and ground truth
        dz = [d_pe(idx) + eta_pos;
              d_ve(idx) + eta_vel];
        dz_total(:,:, j)=dz; %%%%%%%%%%%
        % Initialization
        %
        x0_mean =       [ d_pe(1);
                          d_ve(1);
                  bias_model(1)];
        x0_est = x0_mean + K0*(dz(:,1)-H*x0_mean);
        %
        x_est_total = zeros(3,n_samples);
        x_est_total(:,1) = x0_est;
        x_mean_total = zeros(3,n_samples);
        x_mean_total(:,1) = x0_mean;
        % Matrices
        M_total{1,1} = M0;
        K_total{1,1} = K0;
        P_total{1,1} = P0;
        %%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Real-Time Algorithm
        for i=2:n_samples
            M_prev = M_total{1,i-1};
            K_prev = K_total{1,i-1};
            P_prev = P_total{1,i-1};
            % Equation of Slide 43 - Propagation
            x_mean_total(:,i) = Phi*x_est_total(:, i-1);
            % Verification if this matches the sampling time of the Accelerometer
            if mod(i-1, 40) ~= 0
                 %Fixing the matrixes if that doesn't match
                x_est_total(:, i) = x_mean_total(:,i);
                M_total{1,i} = M_prev;
                K_total{1,i} = K_prev;
                P_total{1,i} = P_prev;
            else
               %Slide 43
                M_next = Phi*P_prev*Phi'+Gamma*W*Gamma';
                K_next = M_next*H'*(H*M_next*H'+V)^(-1);
                P_next = (I_dimKal - K_next*H)*M_next*(I_dimKal - K_next*H)'+ K_next*V*K_next';
                % Update
                M_total{1,i} = M_next;
                K_total{1,i} = K_next;
                P_total{1,i} = P_next;
                x_est_total(:, i) = x_mean_total(:,i) + K_next*(dz(:,(i-1)/40+1) - H*x_mean_total(:,i));

            end
        end
        time(it)=toc;
        % Final vectors
        final_p_est = x_est_total(1,:);
        final_v_est = x_est_total(2,:);
        final_b_est = x_est_total(3,:);
```

```
        final_p_prior  = x_mean_total(1,:);
        final_v_prior  = x_mean_total(2,:);
        final_b_prior  = x_mean_total(3,:);

        for k=1:n_samples
            P_current = P_total{1,k};
             pos_est_var_cell {1,k} = P_current(1,1);
             vel_est_var_cell {1,k} = P_current(2,2);
             bias_est_var_cell {1,k}= P_current(3,3);
        end

        pos_est_var  = cell2mat( pos_est_var_cell );
        vel_est_var  = cell2mat( vel_est_var_cell );
        bias_est_var  = cell2mat( bias_est_var_cell );

        % Priori error
        e_posteriori  = dx_real - x_est_total;

        % Total matrices
        e_posteri_global  (:,:, j)= e_posteriori;% errors
        x_est_global  (:,:, j)= x_est_total;
        x_mean_global(:,:, j)=x_mean_total;
    end
   %  1.Property: Average of the ensemble error
    e_pave = sum(e_posteri_global(1,idx,:) ,  3)/dim;
    e_vave = sum(e_posteri_global(2,idx,:) ,  3)/dim;
    e_bave = sum(e_posteri_global(3,idx,:) ,  3)/dim;

    e_ave = [e_pave;
             e_vave;
            e_bave];

    Prop1p = max(max(e_pave));
    Prop1v = max(max(e_vave));
    Prop1b = max(max(e_bave));

    % 2.Property: (Error Variance vs Filter) +  Orthogonality Properties
    delta  = zeros(3,n_samples_gps,dim);
    P_var_av = zeros(3,3,n_samples_gps);

    err_var  = P_var_av;
    Diff_var  = P_var_av;

    orth  = P_var_av;
    Orth_var = P_var_av;

    for x = 1:n_samples_gps
        for y = 1:dim
        %2.Property: Second Order Stadistics
        delta (:,x,y)  = e_posteri_global (:,x,y)- e_ave(:,x);
        err_var  (:,:, y)  = delta(:,x,y)*delta(:,x,y)';  %Ortogonalidad de los errores

        %3.Property: Independece + orthogonality E[()x']
        orth (:,:, y)  = delta(:,x,y)*x_est_global (:,x,y)';
        end

        %2. Second Order Stadistics
        P_var_av (:,:, x)      =  sum(err_var,3)/(dim-1); %Average del Error Variance
         Diff_var  (:,:, x)     =  P_var_av (:,:, x)  - P_total{1,x}; % Differencia entre  el  Error Variance y el  que te bota el filtro ,  Esto tiene  que ser  zero
```

```
        %3
        Orth_var (:,:, x) =  sum(orth,3)/(dim-1);
    end

    %
    Prop2 = max(max(max(Diff_var)));
    Prop3= max(max(max(Orth_var)));

    % 4.Property:  Residuals

    for  f=1:dim
        res  = dz_total (:,:,  f)  - x_mean_global(1:2, idx,  f);  % medicion del GPS -
        orth_res  (:,:,  f)  = res(:,13+1)*res (:,13)  ';  %  verificar  entre  un tiempo y el tiempo siguiente,  esto  tiene  que ser zero
    end
    Res_ave (:,:)  = (1/dim)*sum(orth_res,3);
    Prop4_res =  max(max(max(Res_ave)));

  %
    %%%%%%%%%%%%%%%%%%
    Flag_validation=1;%
    %%%%%%%%%%%%%%%%%%
    if  Flag_validation==1
        h1 = subplot(3,1,1);
        plot(t_gps,  e_pave)
        hold on
        ylim([-0.3  0.3])
        xlim([0  20]);
        ylabel('average pos error  (m)','interpreter','latex')
        set(gca,'FontSize',font_S)
        grid  on

        h2 =subplot(3,1,2);
        plot(t_gps,  e_vave)
        hold on
        yline (0,  'r--')
        xlim([0  5]);
        ylim([-0.2  0.2])

        ylabel('average  vel  error  (m/s)','interpreter','latex')
        set(gca,'FontSize',font_S)
        grid  on

        h3 =subplot(3,1,3);
        plot(t_gps,  e_bave)
        hold on
        yline (0,  'r--')
        xlim([0  5]);
        ylim([-0.4  0.4])
        ylabel('average bias  error  $(m/s^2)$','interpreter','latex')
        set(gca,'FontSize',font_S)
        grid  on
    end

Final_Table_Properties(it ,:) =[it,  bias_model(1),  dim, Prop1p, Prop1v, Prop1b, Prop2, Prop3 ,Prop4_res];
end
s_legend  = [string(Epochs)+' epochs, '+ '$ t_s= $'+ string(round(time,3))+ ' s',"REF=0"];
subplot(3,1,1);
yline (0,  'r--')
hold on
```

```
legend(h1,s_legend,'interpreter','latex');
subplot(3,1,2);
yline(0, 'r--')
hold on
subplot(3,1,3);
yline(0, 'r--')
hold on
fig = get(groot,'CurrentFigure');
han=axes(fig,'visible','off');
han.XLabel.Visible='On';
xlabel(han,'time (s) (T = 30s)','FontSize',font_S,'interpreter','latex')

X = categorical(string(Final_Table_Properties(:,3)));
%% Verification of Properties
if Flag_validation==1
        fig2 = figure(2)

        h1 = subplot(3,1,1);
        bar(X, Final_Table_Properties(:,7))
        hold on
        title('Error Variance (Simulated vs. Kalman Filter)','interpreter','latex')
        set(gca,'FontSize',font_S)
        grid on

        h2 =subplot(3,1,2);
        bar(X, Final_Table_Properties(:,8))
        hold on
        yline(0, 'r--')
        grid on
        title('Orthogonality (Error vs. Estimated state)','interpreter','latex')
        set(gca,'FontSize',font_S)
        grid on

        h3 =subplot(3,1,3);
        bar(X, Final_Table_Properties(:,9))
        hold on
        yline(0, 'r--')
        title('Residuals Independence','interpreter','latex')
        xlabel('Epochs','FontSize',font_S,'interpreter','latex')
        set(gca,'FontSize',font_S)
        grid on
        xlabel('Epochs(N)','FontSize',font_S,'interpreter','latex')

 end
%%
%[pos_est, vel_est, po, vo]
function [pos_est, vel_est] = dynamics_real(p_mean, p_var, v_mean, v_var, acc_sin,step, n_samples)
%
 p_initial = normrnd(p_mean,sqrt(p_var));
pos_est = zeros(1,n_samples);
pos_est(1) = p_initial;
%
 v_initial = normrnd(v_mean,sqrt(v_var));
 vel_est = zeros(1,n_samples);
 vel_est(1) = v_initial;

    for j=2:n_samples
        vel_est(j) = vel_est(j-1) + acc_sin(j-1)*step;
        pos_est(j) = pos_est(j-1) + vel_est(j-1)*step + 0.5*acc_sin(j-1)*((step))^2;
     end
end
```

```
function [pos_mod, vel_mod] = dynamics_mod(acc_theo, po_mean, vo_mean,step,n_samples)
pos_mod= zeros(1,n_samples); %position
pos_mod(1) = po_mean;
vel_mod = zeros(1,n_samples); %velocity
vel_mod(1) = vo_mean;
    for j=2:n_samples
        vel_mod(j) = vel_mod(j-1) + acc_theo(j-1)*step;
        pos_mod(j) = pos_mod(j-1) + vel_mod(j-1)*step + 0.5*acc_theo(j-1)*(step)^2;
    end
end

function [acc_measu, bias_model, noise_model] = acc_mod(a_sin, noise_mean, noise_var, bias_mean, bias_var,n_samples)
    noise_model = normrnd(noise_mean,sqrt(noise_var),1,n_samples); %Gaussian model, dimension 1 of each element
    bias_model = ones(1,n_samples)*normrnd(bias_mean,sqrt(bias_var));     %The bias is constant
    acc_measu = a_sin + bias_model + noise_model;                          %True model
end


% Is going to measure corrupted measures of the velocity
function [z_pos, z_vel, idx, eta_pos, eta_vel] = GPS_dynamics(n_samples, n_samples_gps, etap_mean, etap_var,
     etav_mean,etav_var, pos_est, vel_est)
    %Index for the update of GPS measurements corresponding to
    %Accelerometer
    idx = 1:40:n_samples;
    %Etas noise
    eta_pos = normrnd(etap_mean,sqrt(etap_var),1,n_samples_gps); % This is not a bias
    eta_vel = normrnd(etav_mean,sqrt(etav_var),1,n_samples_gps); % This is not a bias, random number
    %gps measurements
    z_pos = pos_est(idx) + eta_pos;
    z_vel = vel_est(idx) + eta_vel;
end
```

6.3 Code to plot Figure 2

```
%% Figure 2
T2_flag = 1;
font_S = 20;
if T2_flag ==1
    fig = figure;
    subplot(3,1,1)
    plot(t_sin, pos_est, 'k-',t_sin, pos_mod, 'r--')
    ylabel('position (m)','interpreter','latex')
    legend('Current Position : Integration', 'Position : Measured','interpreter','latex','Location','northwest')
    title('Current vs. Measured Position','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on

    ax_zoom1 = axes('Parent',fig,'Position',[0.77 0.7150 0.1400 0.1000]);
    hold(ax_zoom1,'on'); box(ax_zoom1,'on')
    plot(t_sin, pos_est, 'k-',t_sin, pos_mod, 'r--')
    xlim(ax_zoom1,[21 21.4]);
    ylim(ax_zoom1,[3400 3450]);
    set(gca,'FontSize',font_S)
    grid on

    subplot(3,1,2)
    plot(t_sin, vel_est, 'k-',t_sin, vel_mod, 'r--')
    ylabel('velocity (m/s)','interpreter','latex')
    legend('Current Velocity: Integration', 'Velocity : Measured','interpreter','latex','Location','northwest')
    title('Current vs. Measured velocity','interpreter','latex')
    set(gca,'FontSize',font_S)
```

```
    grid on

    ax_zoom2 = axes('Parent',fig,'Position',[0.77 0.4350 0.1400 0.1000]);
    hold(ax_zoom2,'on'); box(ax_zoom2,'on')
    plot(t_sin, vel_est, 'k-',t_sin, vel_mod, 'r--')
    xlim(ax_zoom2,[21 24]);
    ylim(ax_zoom2,[250 270]);
    set(gca,'FontSize',font_S)
    grid on

    subplot(3,1,3)
    plot(t_sin,acc_sin,'k-',t_sin,acc_measu, 'r--')
    ylabel('acceleration $(m/s^2)$','interpreter','latex')
    legend('Current acceleration', 'Acceleration: Measured','interpreter','latex','Location','northwest')
    title('Current vs. Measured acceleration','interpreter','latex')
    grid on
    set(gca,'FontSize',font_S)
    han=axes(fig,'visible','off');
    han.XLabel.Visible='On';
    xlabel(han,'time (s) (T = 30s)','FontSize',font_S,'interpreter','latex')

    ax_zoom3 = axes('Parent',fig,'Position',[0.60 0.1350 0.1400 0.1000]);
    hold(ax_zoom3,'on'); box(ax_zoom3,'on')
    plot(t_sin, acc_sin, 'k-',t_sin, acc_measu, 'r--')
    xlim(ax_zoom3,[10.5 12]);
    ylim(ax_zoom3,[8.75 9.25]);
    set(gca,'FontSize',font_S)
    grid on

end
```

6.4 Code to plot Figure 3

```
%% Figure 3
% GPS Measurements
Flag_GPS = 1;
if Flag_GPS == 1
    fig =figure;
    subplot(2,2,1)
    plot(t_sin, pos_est,'k-',t_gps, z_pos, 'r--')
    ylabel('position (m)','interpreter','latex')
    legend('Current Position : Integration', 'GPS Position (measured)','interpreter','latex')
    title('Current and GPS Position','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on

    ax_zoom1 = axes('Parent',fig,'Position',[0.18 0.800 0.1400 0.1000]);
    hold(ax_zoom1,'on'); box(ax_zoom1,'on')
    plot(t_sin, pos_est,'k-',t_gps, z_pos, 'r--')
    xlim(ax_zoom1,[17 18]);
    ylim(ax_zoom1,[2490 2530]);
    set(gca,'FontSize',font_S)
    grid on


    subplot(2,2,3)
    plot(t_sin, vel_est,'k-',t_gps, z_vel, 'r--')
    ylabel('velocity (m/s)','interpreter','latex')
    legend('Current Velocity: Integration', 'GPS Velocity (measured)','interpreter','latex')
    title('Current and GPS Velocity','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on
```

```
    ax_zoom2 = axes('Parent',fig,'Position',[0.33 0.1350 0.1400 0.1000]);
    hold(ax_zoom2,'on'); box(ax_zoom2,'on')
    plot(t_sin, vel_est,'k-',t_gps, z_vel, 'r--')
    xlim(ax_zoom2,[18.5 19]);
    ylim(ax_zoom2,[226 233]);
    set(gca,'FontSize',font_S)
    grid on

    subplot(2,2,2)
    scatter(t_gps, eta_pos,'k')
    hold on
    yline(etap_mean, 'b')
    yline(etap_mean + sqrt(etap_var), 'r--')
    yline(etap_mean - sqrt(etap_var), 'r--')
    ylabel('position noise (m)','interpreter','latex')
    legend('GPS Position Noise', '$w$ Mean','$\pm \sigma$','interpreter','latex','Location','southwest')
    title('GPS Measurement Noise: Position','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on

    subplot(2,2,4)
    scatter(t_gps, eta_vel,'k')
    hold on
    yline(etav_mean, 'b')
    yline(etav_mean + sqrt(etav_var), 'r--')
    yline(etav_mean - sqrt(etav_var), 'r--')
    ylabel('velocity noise m/s','interpreter','latex')
    legend('GPS Velocity Noise','$w$ Mean','$\pm \sigma$','interpreter','latex','Location','southwest')
    title('GPS Measurement Noise: Velocity','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on

    han=axes(fig,'visible','off');
    han.XLabel.Visible='On';
    xlabel(han,'time (s) (T = 30s)','FontSize',font_S,'interpreter','latex')
end
```

6.5 Code to plot Figure 4

```
%% Figure 4
font_S = 20;
Flag_plot_delta_all_approach = 1;
if Flag_plot_delta_all_approach ==1
    fig = figure;
    subplot(2,1,1)
    plot(t_sin, d_pe, '-',t_gps, dz(1,:), '--',t_gps, pos_est(idx) - z_pos, '--')
    ylabel('position (m)','interpreter','latex')
    legend('$\Delta$ Current pos / Accelerometer Position', ...
        '$\Delta$ GPS pos / Accelerometer Position', ...
        '$\Delta$ Current pos / GPS Position','interpreter','latex')
    title(' Comparison of $\Delta$ of Position: 3 models','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on

    subplot(2,1,2)
    plot(t_sin, d_ve, '-',t_gps, dz(2,:), '--',t_gps, vel_est(idx) - z_vel, '--')
    ylabel('velocity (m/s)','interpreter','latex')
    legend('$\Delta$ Current vel / Accelerometer Velocity', ...
        '$\Delta$ GPS vel / Accelerometer Velocity', ...
        '$\Delta$ Current vel / GPS Velocity','interpreter','latex')
    title('Comparison of $\Delta$ of Velocity: 3 models','interpreter','latex')
```

```
    set(gca,'FontSize',font_S)
     grid on

    han=axes(fig,'visible','off');
    han.XLabel.Visible='On';
    xlabel(han,'time (s) (T = 30s)','FontSize',font_S,'interpreter','latex')

end
```

6.6 Code to plot Figures 5, 6, and 7

```
%% Figures 5-7
%KF Estimates: Accelerometer Time D!
font_S = 20;
Flag_Kalman = 1;
 if Flag_Kalman ==1
    clc
    close all
    fig1 = figure;
    subplot(3,1,1)
    plot(t_sin,d_pe,'-',t_sin, final_p_est ,'k--',t_sin, final_p_est + sqrt(pos_est_var),'r-',t_sin, final_p_est - sqrt(
     pos_est_var), 'r-')
    ylabel('position (m)','interpreter','latex')
    legend({'$\Delta$ Current pos/ Accelerometer pos', 'Estimated $\Delta$ Position', '$\pm \sigma$'},'interpreter','
     latex','Location','southwest');
     title ('Current vs Estimated $\Delta$ Position','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on

    ax_zoom1 = axes('Parent',fig1,'Position',[0.75 0.83 0.1400 0.1000]);
    hold(ax_zoom1,'on'); box(ax_zoom1,'on')
     plot(t_sin,d_pe,'-',t_sin, final_p_est ,'k--',t_sin, final_p_est + sqrt(pos_est_var),'r-',t_sin, final_p_est - sqrt(
     pos_est_var), 'r-')
    ylim(ax_zoom1,[-16.5 -15]);
    xlim(ax_zoom1,[14.8 15.2]);
    set(gca,'FontSize',font_S)
    grid on

    subplot(3,1,2)
    plot(t_sin,d_ve,'-',t_sin, final_v_est ,'k--',t_sin, final_v_est + sqrt( vel_est_var ), 'r-',t_sin, final_v_est - sqrt(
     vel_est_var ), 'r-')
    ylabel('velocity (m/s)','interpreter','latex')
    legend('$\Delta$ Current vel/ Accelerometer vel', 'Estimated $\Delta$ Velocity', '$\pm \sigma$','interpreter','latex
     ','Location','southwest')
     title ('Current vs Estimated $\Delta$ Velocity','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on

    ax_zoom2 = axes('Parent',fig1,'Position',[0.75 0.55 0.1400 0.1000]);
    hold(ax_zoom2,'on'); box(ax_zoom2,'on')
     plot(t_sin,d_ve,'-',t_sin, final_v_est ,'k--',t_sin, final_v_est + sqrt( vel_est_var ), 'r-',t_sin, final_v_est - sqrt(
     vel_est_var ), 'r-')
    xlim(ax_zoom2,[16 17]);
    ylim(ax_zoom2,[-2.75 -2.5]);
    set(gca,'FontSize',font_S)
    grid on

    subplot(3,1,3)
    plot(t_sin, bias_model, '-',t_sin, final_b_est , 'k--',t_sin, final_b_est + sqrt( bias_est_var ), 'r-',t_sin, final_b_est
      - sqrt( bias_est_var ), 'r-')
    ylabel('bias ($m/s^2$)','interpreter','latex')
    legend('Real Bias', 'Estimated Bias','$\pm \sigma$','interpreter','latex','Location','southwest')
```

```
title('Current vs. Estimated Bias','interpreter','latex')
set(gca,'FontSize',font_S)
grid on

han=axes(fig1,'visible','off');
han.XLabel.Visible='On';
xlabel(han,'time (s) (T = 30s)','interpreter','latex')
set(gca,'FontSize',font_S)

set(fig1, 'Renderer', 'painters');

%Variances
fig2 = figure;
subplot(3,1,1)
plot(t_sin, pos_est_var, 'k-')
ylabel('pos variance $(m^2)$','interpreter','latex')
title('$\Delta$ Position Variance','interpreter','latex')
set(gca,'FontSize',font_S)
grid on

subplot(3,1,2)
plot(t_sin, vel_est_var, 'k-')
ylabel('vel variance $(m/s)^2$','interpreter','latex')
title('$\Delta$ Velocity Variance','interpreter','latex')
set(gca,'FontSize',font_S)
grid on

subplot(3,1,3)
plot(t_sin, bias_est_var, 'k-')
ylabel('bias variance $(m/s^2)^2$','interpreter','latex')
title('Bias Variance','interpreter','latex')
set(gca,'FontSize',font_S)
grid on

han=axes(fig2,'visible','off');
han.XLabel.Visible='On';
xlabel(han,'time (s) (T = 30s)','interpreter','latex')
set(gca,'FontSize',font_S)
grid on

%Error
fig3 = figure;
subplot(3,1,1)
plot(t_sin,d_pe - final_p_est,'k-',t_sin,d_pe-final_p_est + sqrt(pos_est_var),'r-',t_sin,d_pe-final_p_est - sqrt(pos_est_var), 'r-')
yline(bias_model(1), 'b')
ylabel('position m','interpreter','latex')
legend('position $\Delta$ error', '$\pm \sigma$','interpreter','latex','Location','southwest')
title('$\Delta$ Position Error','interpreter','latex')
set(gca,'FontSize',font_S)
grid on

subplot(3,1,2)
plot(t_sin, d_ve - final_v_est, 'k-',t_sin,d_ve - final_v_est + sqrt(vel_est_var), 'r-',t_sin,d_ve - final_v_est - sqrt(vel_est_var), 'r-')
yline(0, 'b')
ylabel('velocity m/s','interpreter','latex')
legend('velocity $\Delta$ error','$\pm \sigma$','interpreter','latex','Location','southwest')
title('$\Delta$ Velocity Error','interpreter','latex')
set(gca,'FontSize',font_S)
grid on
```

```
    subplot(3,1,3)
    plot(t_sin, bias_model - final_b_est, 'k-',t_sin,bias_model - final_b_est + sqrt(bias_est_var), 'r-',t_sin,
     bias_model - final_b_est - sqrt(bias_est_var), 'r-')
    yline(0, 'b')
    ylabel('bias $(m/s^2)$','interpreter','latex')
    legend('bias error','$\pm \sigma$','interpreter','latex','Location','southwest')
     title('Bias Error','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on

    han=axes(fig3,'visible','off');
    han.XLabel.Visible='On';
    xlabel(han,'time (s)','FontSize',font_S,'interpreter','latex')
end
```

6.7 Code to plot Figure 8

```
%% Figure 8
%Errors - One Realization D!
%GPS time
font_S = 20;
Flag_Errors_T1 =1;
 if Flag_Errors_T1 ==1
    fig1 = figure;
    ax_zoom2 = subplot(3,1,1);
    plot(t_gps, e_posteriori (1,idx),'k--',t_gps, e_priori (1,idx),'r--')
    yline(0, 'b')
    ylabel('position (m)','interpreter','latex')
    legend('apriori Position error', 'aposteriori Position error','interpreter','latex')
     title('apriori-posteriori Error Comparison: $\Delta$ Position','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on

    ax_zoom2 = axes('Parent',fig1,'Position',[0.5 0.73 0.1400 0.1000]);
    hold(ax_zoom2,'on'); box(ax_zoom2,'on')
     plot(t_gps, e_posteriori (1,idx),'k--',t_gps, e_priori (1,idx),'r--')
    ylim(ax_zoom2,[-0.05 0.05]);
    xlim(ax_zoom2,[21 23]);
    set(gca,'FontSize',font_S)
    grid on


    subplot(3,1,2)
    plot(t_gps, e_posteriori (2,idx),'k--',t_gps,e_priori (2,idx),'r--')
    yline(0, 'b')
    ylabel('velocity (m/s)','interpreter','latex')
    legend('apriori Velocity error', 'aposteriori Velocity error','interpreter','latex')
     title('apriori-posteriori Error Comparison: $\Delta$ Velocity','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on

    subplot(3,1,3)
    plot(t_gps, e_posteriori (3,idx),'k--',t_gps,e_priori (3,idx),'r--')
    yline(0, 'b')
    ylabel('bias $(m/s^2)$','interpreter','latex')
    legend('prior bias error', 'bias posterior error','interpreter','latex')
     title('arior-posteriori Error Comparison: Bias','interpreter','latex')
    set(gca,'FontSize',font_S)
    grid on

    han=axes(fig1,'visible','off');
```

```
    han.XLabel.Visible='On';
    xlabel(han,'time (s)','FontSize',font_S,'interpreter','latex')
end
```

References

[1] Mateos, G. (2019). Gaussian, markov and stationary processes. https://www.hajim.rochester.edu/ece/sites/gmateos/ECE440/Slides/block_5_stationary_processes_part_a.pdf.

[2] Penn State, E. C. o. S. (2004). Conditional probability. https://online.stat.psu.edu/stat414/lesson/19/19.2.

[3] SBS-Systems (2022). Imu-inertial measurement unit. https://www.sbg-systems.com/inertial-measurement-unit-imu-sensor/.

[4] Yale, S. (1998). Conditional probability. http://www.stat.yale.edu/Courses/1997-98/101/condprob.htm.