MAE 271A Project
Calibration of an Accelerometer Using GPS Measurements

Zhiyuan Cao

December 2, 2015

Abstract

INS/IMU (Inertial Navigation System/Inertial Measurement Unit) are widely used to measure position and velocity of a vehicle in an inertial space. However such measurement is usually accompanied by GPS signal for calibration. The measuring process is in essence a stochastic process, meaning error and noise will be introduced and contaminate the data. In this report, such measure and calibration process will be modeled and a Kalman filter will be implemented. Based on the measurement from the difference between the GPS and IMU, the estimator will give out estimation of the GPS signal. The estimator is validated through several approaches including orthogonal checks to evaluate its performance. The validation proves the implementation of estimator to be successful.

Key Words: GPS, IMU, Stochastic Analysis, Kalman Filter

Introduction

The objective of this project is to estimate the one dimensional position and velocity of a vehicle given two data sources:

1. GPS
2. Accelerometer

The data from the accelerometer is integrated to produce the velocity and position of the vehicle. It could provide an accurate measurement in a short period. However, since the INS accelerometers produce an unknown bias signal which would be integrated twice and produce error in velocity and position, a GPS satellite signal is used to correct or calibrate the measurement from the INS/IMU. The advantages to use INS/IMU with GPS are that the INS/IMU can be calibrated by GPS and that the INS/IMU can provide velocity and position at a faster sampling rate. Also, GPS may lose signals and the dynamics of the vehicle could still be computed through INS/IMU.

In this project, the difference of the GPS signal and INS/IMU is used as the measurement to estimate the GPS signal. The system diagram is as followed:

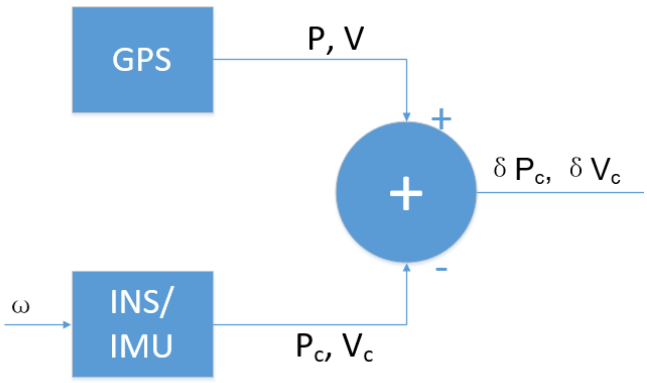

Fig. 1 System Diagram

The report will be organized as follow: first the model of the GPS and the accelerometer will be introduced. Then the Kalman filter will be implemented and performance will be evaluated and discussed, followed by the Conclusion. The Matlab code is in Appendix.

Model Formation

Truth Model

The truth model being used as the simulation model is as following:

$$a(\mathrm{t}) = A\sin(2\pi\omega\,\mathrm{t})$$

Where a and ω are given values. Through integration we have the velocity v and position p as following:

$$v(\mathrm{t}) = v(0) + \frac{a}{2\pi\omega} - \frac{a}{2\pi\omega}\cos(2\pi\omega\,\mathrm{t})$$

$$p(\mathrm{t}) = p(0) + (v(0) + \frac{a}{2\pi\omega})t - \frac{a}{4\pi^2\omega^2}\sin(2\pi\omega\,\mathrm{t}) \qquad (1)$$

Where

$$v(0) \sim \mathrm{N}(\bar{v}_0, \mathrm{M}_0^v)$$

$$p(0) \sim \mathrm{N}(\bar{p}_0, \mathrm{M}_0^p)$$

Accelerometer Model

The 200Hz accelerometer model used in this project is as following:

$$a_c(\mathrm{t_j}) = a(\mathrm{t_j}) + \mathrm{b} + \omega(\mathrm{t_j})$$

Where

$$\mathrm{t_j} - \mathrm{t_{j+1}} = \triangle\mathrm{t} = 0.005\,\mathrm{sec}.$$

Therefore, the velocity v and position p:

$$v_c(\mathrm{t_{j+1}}) = v_c(\mathrm{t_j}) + a_c(\mathrm{t_j})\triangle\mathrm{t}$$

$$p_c(\mathrm{t_{j+1}}) = p_c(\mathrm{t_j}) + v_c(\mathrm{t_j})\triangle\mathrm{t} + a_c(\mathrm{t_j})\frac{\triangle\mathrm{t}^2}{2} \qquad (2)$$

Where

$$v_c(0) = \bar{v}_0$$

$$p_c(0) = \bar{p}_0 = 0$$

Derivation of the Dynamic Model

To have a system model independent of the actual acceleration, assume the true acceleration is integrated the same way given in (2)

$$v_E(t_{j+1}) = v_E(t_j) + a_E(t_j)\Delta t$$

$$p_E(t_{j+1}) = p_E(t_j) + v_E(t_j)\Delta t + a(t_j)\frac{\Delta t^2}{2} \tag{3}$$

Subtract (3) from (2) and the dynamics of the system is as follow:

$$\begin{bmatrix} \delta p_E(t_{j+1}) \\ \delta v_E(t_{j+1}) \\ b(t_{j+1}) \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & -\frac{\Delta t^2}{2} \\ 0 & 1 & -\Delta t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \delta p_E(t_j) \\ \delta v_E(t_j) \\ b(t_j) \end{bmatrix} - \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \\ 0 \end{bmatrix} \omega(t_j) \tag{4}$$

Where

$$\delta p_E(t_0) = p_E(t_0) - p_c(t_0) \sim N(0, M_0^p)$$

$$\delta v_E(t_0) = v_E(t_0) - v_c(t_0) \sim N(0, M_0^v)$$

$$E[\omega(t_j)] = 0, E[\omega(t_j)\omega(t_l)^T] = W\delta_{j,l}$$

$$b(0) \sim N(0, M_0^b)$$

This concludes that the model coefficients are independent of the acceleration profile.

Measurement Equations

The GPS measurement has a sample rate of 5Hz and is corrupted by a white noise with the following form:

$$z^p(t_i) = p(t_i) + \eta^p(t_i)$$

$$z^v(t_i) = v(t_i) + \eta^v(t_i) \tag{5}$$

Where

$$t_i - t_{i+1} = 40\Delta t = 0.2\sec .$$

$$\eta^p(t_i) \sim N(0,1)$$

$$\eta^v(t_i) \sim N(0, 0.04)$$

To construct the Kalman filter, subtract (2) from (6) and we have

$$\delta z^p(t_i) = \delta p(t_i) + \eta^p(t_i)$$

$$\delta z^v(t_i) = \delta v(t_i) + \eta^v(t_i) \tag{6}$$

Where

$$\delta p(t_i) = \delta p(t_i) - \delta p_c(t_i)$$

$$\delta v(\mathrm{t}_i) = \delta v(\mathrm{t}_i) - \delta v_c(\mathrm{t}_i)\,.$$

Kalman Filter

If we assume the approximation could be made that $\delta p(\mathrm{t}_i) = \delta p_E(\mathrm{t}_i)$ and $\delta v(\mathrm{t}_i) = \delta v_E(\mathrm{t}_i)$, based on the measurement equation (6) and the dynamics (4), the approximate posteriori conditional mean $\delta x(\mathrm{t}_i)$ and the conditional posteriori error variance $P(\mathrm{t}_i)$ can be computed through Kalman filter algorithm if we define

$$\delta x(\mathrm{t}_i) = \begin{bmatrix} \delta p(\mathrm{t}_i) \\ \delta v(\mathrm{t}_i) \\ \hat{b}(\mathrm{t}_i) \end{bmatrix} = \begin{bmatrix} p(\mathrm{t}_i) - p(\mathrm{t}_i) \\ \hat{v}(\mathrm{t}_i) - v(\mathrm{t}_i) \\ \hat{b}(\mathrm{t}_i) \end{bmatrix}$$

The pseudo-code or the Kalman filter algorithm is given below:

IF(New GPS Measurement Comes)

$K_k = M_k H^T (\mathrm{HM_kH^T + V})^{-1}$ % Update Kalman Gain

$x_k = \bar{x}_k + K_k(\mathrm{z_k} - H\bar{x}_k)$ % Update Conditional Mean

$P_k = (M_k^{-1} + H^T V^{-1} H)^{-1}$ % Update Conditional Variance

$\bar{x}_{k+1} = \Phi x_k$ % Propagate the mean

$M_{k+1} = \Phi P_k \Phi^T + \Gamma V^{-1} \Gamma^T$ % Propagate the variance

END

Results

Single Realization

Fig. 2 shows the implementation of Kalman filter for a single run. In the figure, the estimated position, velocity and bias of the system is shown. In the figure, the red "+" is the estimate of GPS signal and the blue line is the truth value. As we can see in the Figure, since the difference between real position and estimated position is too small compared to their value, we cannot get a very clear idea of how the filter has been doing. For bias although the difference between estimation and real value is obvious, it quickly converge after 5 seconds.

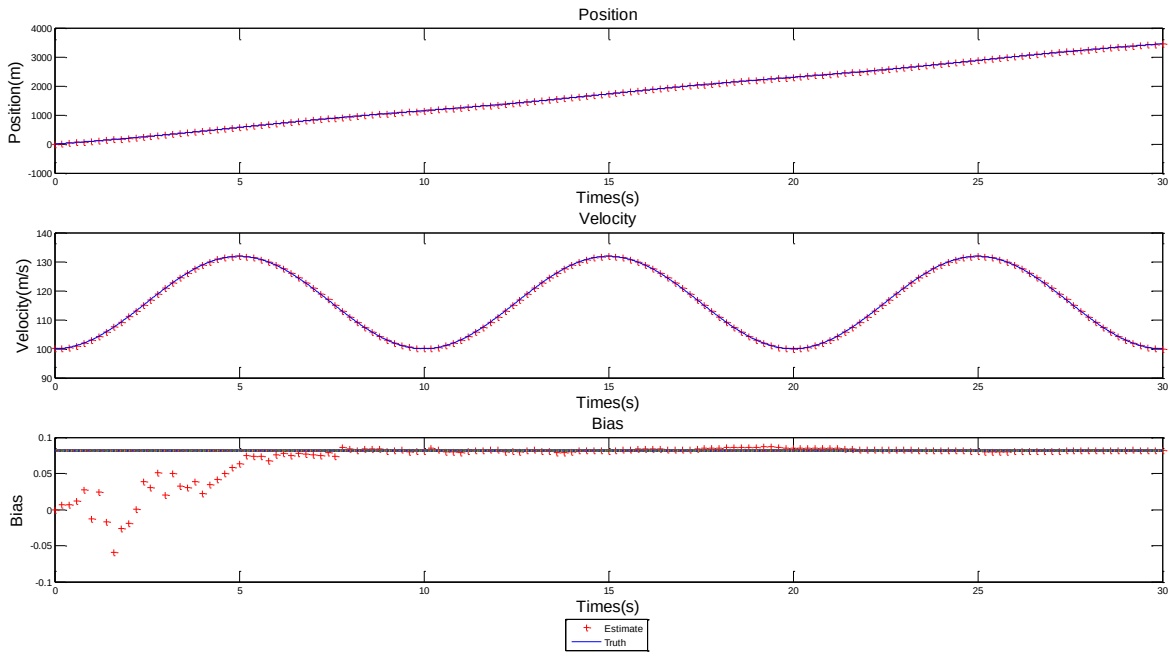

Figure 2: Estimates from a Single Run

Figure 3 provides some ideas of the Kalman filter performance. It shows the error in a single run compared to its one sigma bound. For one single run, it is possible that a part of the error falls out of the one sigma bound as shown in Figure 3.

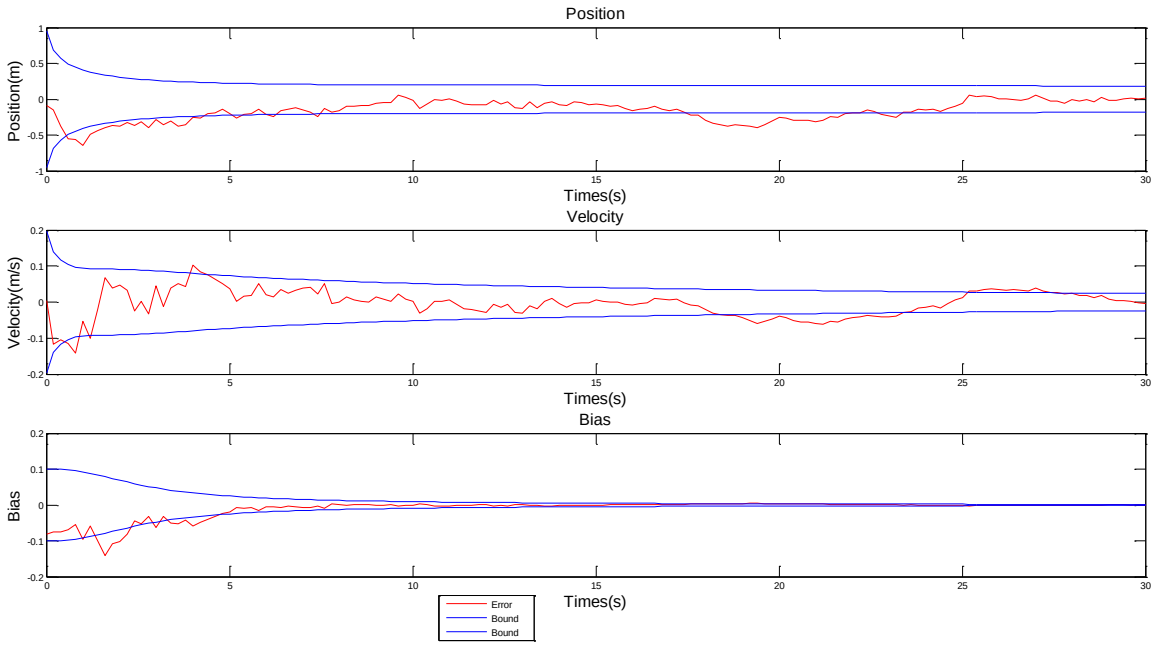

Figure 3: Error in Estimates from a Single Run

Validation and Verification over a set of realizations

Validation and verification will be shown in four aspects

Actual Error

Define $\delta x(\mathrm{t}_j)$ as following:

$$\delta x(\mathrm{t}_j)=\begin{bmatrix}\delta p(\mathrm{t}_j)\\ \delta v(\mathrm{t}_j)\\ b(\mathrm{t}_j)\end{bmatrix}=\begin{bmatrix}p(\mathrm{t}_j)-p_c(\mathrm{t}_j)\\ v(\mathrm{t}_j)-v_c(\mathrm{t}_j)\\ b(\mathrm{t}_j)\end{bmatrix}$$

Where $p(\mathrm{t}_j)$, $v(\mathrm{t}_j)$ and $b(\mathrm{t}_j)$ are given from the truth model. From here we could define the actual a priori estimation error as:

$$\bar{\mathrm{e}}(\mathrm{t}_j)=\delta x(\mathrm{t}_j)-\overline{\delta x}(\mathrm{t}_j)=\begin{bmatrix}p(\mathrm{t}_j)-p_c(\mathrm{t}_j)\\ v(\mathrm{t}_j)-v_c(\mathrm{t}_j)\\ b(\mathrm{t}_j)\end{bmatrix}-\begin{bmatrix}\bar{\mathrm{p}}(\mathrm{t}_j)-p(\mathrm{t}_j)\\ \bar{\mathrm{v}}(\mathrm{t}_j)-v(\mathrm{t}_j)\\ \bar{\mathrm{b}}(\mathrm{t}_j)\end{bmatrix}=\begin{bmatrix}p(\mathrm{t}_j)-\bar{\mathrm{p}}(\mathrm{t}_j)\\ v(\mathrm{t}_j)-\bar{\mathrm{v}}(\mathrm{t}_j)\\ b(\mathrm{t}_j)-\bar{\mathrm{b}}(\mathrm{t}_j)\end{bmatrix}$$

The posteriori estimation error is defined similarly as $e(\mathrm{t}_j)=\delta x(\mathrm{t}_j)-\delta x(\mathrm{t}_i)$. At measurement time $t_i=t_j$, $\bar{\mathrm{e}}(\mathrm{t}_j)=\delta x(\mathrm{t}_j)-\overline{\delta x}(\mathrm{t}_j^-)$ and the posteriori estimation error is $e(\mathrm{t}_j)=\delta x(\mathrm{t}_j)-\delta x(\mathrm{t}_i^+)$.

Let $e^l(\mathrm{t}_\mathrm{i})$ represent the actual error for realization *l*.

The ensemble average of $e^l(\mathrm{t}_\mathrm{i})$ which produce the actual mean is:

$$e^{ave}(\mathrm{t}_\mathrm{i})=\frac{1}{N_{ave}}\sum_{l=1}^{N_{ave}}e^l(\mathrm{t}_\mathrm{i})$$

N_{ave} represents the number of realizations. It should be large enough to produce good approximation. In this project, N_{ave} = 10000. It is expected that $e^{ave}(\mathrm{t}_\mathrm{i})\approx 0$ for all $\mathrm{t}_\mathrm{i}\in[0,30]$. And the result of this validation is shown in Fig. 4. As we can see, the average is very close to zero.

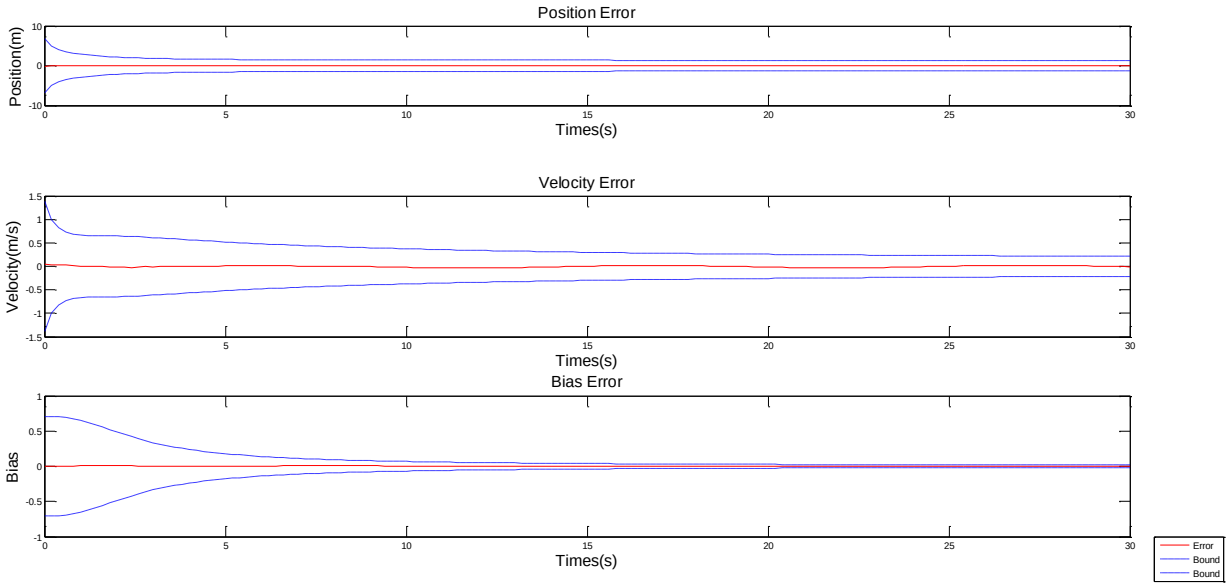

Fig. 4 Error Average over an ensemble of realizations

Actual Error Variance

The actual error variance $P^{ave}(\mathrm{t_i})$ can be calculated as:

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

Here we used N_{ave}-1 instead of N_{ave} because we applied the small sample theory for an unbiased variance. The matrix $P^{ave}(\mathrm{t_i})$ should be close to $P(\mathrm{t_i})$ computed in the Kalman filter algorithm. The result of actual error variance check (difference between is $P^{ave}(\mathrm{t_i})$ and $P(\mathrm{t_i})$) is shown in Fig. 5. For all nine elements in the matrix, the difference swiftly converge to zero.

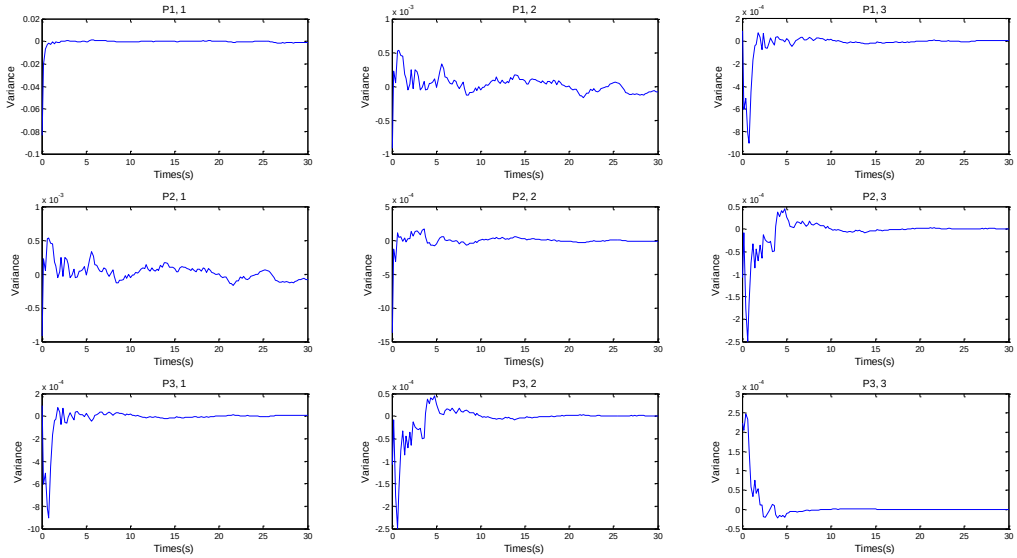

Fig. 5 Difference between actual error variance and propagated covariance

Orthogonality of the error

Similarly, the orthogonality of the error is checked by the following average:

$$\frac{1}{N_{ave}}\sum_{l=1}^{N_{ave}}\left[e^{l}(\mathrm{t_i})-e^{ave}(\mathrm{t_i})\right]x(\mathrm{t}_i)^{\mathrm{T}}\approx 0\forall t_i$$

The result is shown in Fig. 6, all nine elements is close to zero.

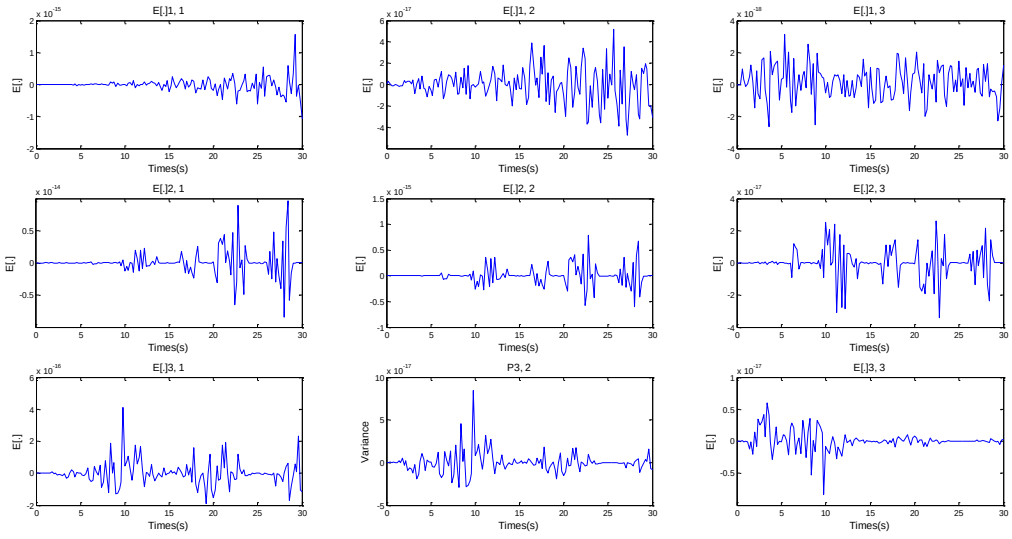

Fig. 6 Orthogonality of error against the estimate

Residual

Finally the independence of the residuals also need to be checked. The residual for one realization *l* is

$$r^l(\mathrm{t_i}) = \begin{bmatrix} \delta z^{pl}(\mathrm{t}_i) - \overline{\delta z}^{pl}(\mathrm{t}_i) \\ \delta z^{vl}(\mathrm{t}_i) - \overline{\delta z}^{vl}(\mathrm{t}_i) \end{bmatrix}$$

Thus, the ensemble average for the correlation of the residuals is:

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

For residual check we only have to validate one value of t_m and t_i.

The result is shown in (7)

$$\frac{1}{N_{ave}} \sum_{l=1}^{N_{ave}} r^l(t_i) r^l(t_m)^{\mathrm{T}} = \begin{bmatrix} 0.0055 & 0.0077 \\ 0.0018 & 0.0006 \end{bmatrix}$$

This is also close to zero as expected. The above checks are to be called Monte Carlo Analysis.

Conclusion

The model of GPS calibrating IMU/INS is formed and the implementation of Kalman to gain the estimation is proven to be successful according to the Monte Carlo Analysis. Through such process, I gained a much better understanding on the usefulness of estimation to tackle with real world problems and how Kalman filter is subtracted, modeled and implemented.

Appendix

Single Realization

```
%--------------------------------------------------------------------------
%--------------------------------------------------------------------------
% MAE 271 Project
% Zhiyuan Cao
% 304397496
% KalmanSingle.m
%--------------------------------------------------------------------------
clc;clear;
%--------------------------------------------------------------------------
% Initial Values
%--------------------------------------------------------------------------
A = 10;
om = 2*pi*0.1;
freqAcc = 200;
dt = 1/freqAcc;
freqGps = 5;
dtGps = 1/freqGps;
freqDiff = dtGps/dt;
T = 30;
tIMU = 0 : dt : T;
tGps = 0 : dtGps : T;
NOM = length(tGps); % Number of measurement
NOS = length(tIMU); % Number of Sample
%--------------------------------------------------------------------------
% Stats
%--------------------------------------------------------------------------
veloMu = 100;
posiMu = 0;
biasMu = 0;
noiseMu = 0;
gpsCorrMu = [0; 0];

veloSigma = 1;
posiSigma = 10;
biasSigma = .01;
noiseSigma = .0004;
gpsCorrSigma = [1, 0; 0, .04];
%--------------------------------------------------------------------------
% noise and Bias
%--------------------------------------------------------------------------
gpsCorr = zeros(2, NOS);
gpsCorr(1, :)= gpsCorrMu(1) + sqrt(gpsCorrSigma(1))*randn(1, NOS);
gpsCorr(2, :)= gpsCorrMu(2) + sqrt(gpsCorrSigma(4))*randn(1, NOS);
bias = normrnd(biasMu, biasSigma);
noise = noiseMu + sqrt(noiseSigma)*randn(1, NOS);
%--------------------------------------------------------------------------
% Real values
%--------------------------------------------------------------------------
accReal = A*sin(om*tIMU);
veloZero = normrnd(veloMu, veloSigma);
posiZero = normrnd(posiMu, posiSigma);
veloReal = veloZero + A/om*(1 - cos(om*tIMU));
posiReal = posiZero + (veloZero + A/om)*tIMU - A*sin(om*tIMU)/om^2;
```

```
%--------------------------------------------------------------------------
% Acc values
%--------------------------------------------------------------------------
veloAcc = zeros(1, NOS);
posiAcc = zeros(1, NOS);

accAcc = accReal + bias + noise;
veloAcc(1) = veloMu;
posiAcc(1) = posiMu;
for Iter = 2 : NOS
    veloAcc(Iter) = veloAcc(Iter - 1) + accAcc(Iter - 1)*dt;
    posiAcc(Iter) = posiAcc(Iter - 1) + veloAcc(Iter - 1)*dt ...
        + accAcc(Iter - 1)*dt^2/2;
end
%--------------------------------------------------------------------------
% One Realization
%--------------------------------------------------------------------------
posiDiff = posiReal(1 : freqDiff : end) - posiAcc(1 : freqDiff : end);
veloDiff = veloReal(1 : freqDiff : end) - veloAcc(1 : freqDiff : end);
z(1, :) = posiDiff + gpsCorr(1, 1 : freqDiff : end);
z(2, :) = veloDiff + gpsCorr(2, 1 : freqDiff : end);
%--------------------------------------------------------------------------
% Dynamics
%--------------------------------------------------------------------------
phi = [1, dtGps, -dtGps^2/2; 0, 1, -dtGps; 0, 0, 1];
phiT = transpose(phi);
h = [1, 0, 0; 0, 1, 0];
hT = transpose(h);
gamma = -[dtGps^2/2; dtGps; 0];
gammamaT = transpose(gamma);
mIter = [posiSigma, 0, 0; 0, veloSigma, 0; 0, 0, biasSigma];
xbar = [0; 0; 0];
%--------------------------------------------------------------------------
% Kalman Filter Implement
%--------------------------------------------------------------------------
xHat = zeros(3, NOM);
xBar = zeros(3, NOS);
est = zeros(3, NOM);
pStore = zeros(3, 3, NOM);
pDiag = zeros(3, NOM);
res = zeros(2, NOM);

Itergps = 1;
for Iter = 1:NOM
    %------------------------------
    % Update Conditional Mean
    %------------------------------
    K = mIter*hT/(h*mIter*hT + gpsCorrSigma);
    residual = (z(:, Itergps) - h*xbar);
    xhat = xbar + K*residual;
    res(:, Itergps) = residual;
    %------------------------------
    % Update Conditional Variance
    %------------------------------
    pIter = (eye(3) - K*h)*mIter*transpose((eye(3) - K*h)) +...
        K * gpsCorrSigma * transpose(K);
```

```
    %------------------------------
    % Propagate the mean
    %------------------------------
    xbar = phi*xhat;
    xBar(:, Iter) = xbar;
    %------------------------------
    % Propagate the variance
    %------------------------------
    mIter = phi*pIter*phiT + gamma*noiseSigma*gammamaT;

    xHat(:, Itergps) = xhat;
    pStore(:, :, Itergps) = pIter;
    Itergps = Itergps + 1;

end
est(1, :) = xHat(1, :) + posiAcc(1 : freqDiff : end);
est(2, :) = xHat(2, :) + veloAcc(1 : freqDiff : end);
est(3, :) = xHat(3, :);
err(1, :) = est(1, :) - posiReal(1 : freqDiff : end);
err(2, :) = est(2, :) - veloReal(1 : freqDiff : end);
err(3, :) = est(3, :) - bias(1 : freqDiff : end);
%--------------------------------------------------------------------------
% Plot
%--------------------------------------------------------------------------
set(0,'defaultfigurecolor',[1 1 1]);
figure(1)
subplot(3,1,1)
plot(tGps, est(1, :), 'r+'), hold on
plot(tIMU, posiReal(1, :))
title('Position', 'FontSize', 16)
xlabel('Times(s)', 'FontSize', 16)
ylabel('Position(m)', 'FontSize', 16);

subplot(3,1,2)
plot(tGps, est(2, :), 'r+'), hold on
plot(tIMU, veloReal(1, :))
title('Velocity', 'FontSize', 16)
xlabel('Times(s)', 'FontSize', 16)
ylabel('Velocity(m/s)', 'FontSize', 16);


subplot(3,1,3)
plot(tGps, est(3, :), 'r+'), hold on
plot(tIMU, bias)
title('Bias', 'FontSize', 16)
xlabel('Times(s)', 'FontSize', 16)
ylabel('Bias', 'FontSize', 16);

figure(2)
subplot(3,1,1)
plot(tGps, err(1, :), 'r'), hold on
plot(tGps, sqrt(pDiag(1, :))), hold on
plot(tGps, -sqrt(pDiag(1, :)))
title('Position', 'FontSize', 16)
xlabel('Times(s)', 'FontSize', 16)
ylabel('Position(m)', 'FontSize', 16);
```

```
subplot(3,1,2)
plot(tGps, err(2, :), 'r'), hold on
plot(tGps, sqrt(pDiag(2, :))), hold on
plot(tGps, -sqrt(pDiag(2, :)))
title('Velocity', 'FontSize', 16)
xlabel('Times(s)', 'FontSize', 16)
ylabel('Velocity(m/s)', 'FontSize', 16);


subplot(3,1,3)
plot(tGps, err(3, :), 'r'), hold on
plot(tGps, sqrt(pDiag(3, :))), hold on
plot(tGps, -sqrt(pDiag(3, :)))
title('Bias', 'FontSize', 16)
xlabel('Times(s)', 'FontSize', 16)
ylabel('Bias', 'FontSize', 16);
```

Validation over an ensemble of realizations

```
%--------------------------------------------------------------------------
% MAE 271
% Zhiyuan Cao
% 304397496
%--------------------------------------------------------------------------
clc; clear;

n=10000; % Number of Ensemble
NOM = 151;

errAver = zeros(3, NOM);
errStore = zeros(3, NOM, n);
pAver = zeros(3, 3, NOM);
pAverDiag = zeros(3, NOM);
pStoreAver = zeros(3, 3, NOM);
orthEstErr= zeros(3, 3, NOM);
orthRes = zeros(2, 2);
%--------------------------------------------------------------------------
% Run and Collect Data
%--------------------------------------------------------------------------
printCount = 1;
for Iter1 = 1 : n
    KalmanMain;

    errStore(:, :, Iter1) = err;
    errAver = err + errAver;
    pAver = pAver + pStore;
    pAverDiag = pAverDiag + pDiag;
    if printCount == 100
        disp(Iter1)
        printCount = 0;
    end
    printCount = printCount + 1;
end
%--------------------------------------------------------------------------
% Checks
```

```
%-------------------------------------------------------------------------
errAver = errAver/n; % Ensemble Average
pAver = pAver/n; % Ensemble Average Variance
for Iter1 = 1 : n
    for Iter2 = 1 : NOM
        errDiff = errStore(:, Iter2, Iter1)-errAver(:, Iter2);
        pStoreAver(:, :, Iter2) = pStoreAver(:, :, Iter2) +...
 (errDiff) * transpose(errDiff);
        orthEstErr(:, :, Iter2) = orthEstErr(:, :, Iter2) +...
 errDiff * transpose(xHat(:, Iter2));
    end
    orthRes(:, :) = orthRes(:, :) + res(:, 30)*transpose(res(:, 20));
end
pStoreAver = pStoreAver/(n-1); % Actual error Variance
orthEstErr = orthEstErr/n; % Orthogonality of the Error
orthRes = orthRes/n; % Independence of the residuals
%-------------------------------------------------------------------------
% Plot
%-------------------------------------------------------------------------
set(0,'defaultfigurecolor',[1 1 1]);
figure(1)
subplot(3,1,1)
plot(tGps, errAver(1, :), 'r'), hold on
plot(tGps, sqrt(pAverDiag(1, :)), '--'), hold on
plot(tGps, -sqrt(pAverDiag(1, :)), '--')
title('Position Error', 'FontSize', 16)
xlabel('Times(s)', 'FontSize', 16)
ylabel('Position(m)', 'FontSize', 16);
subplot(3,1,2)
plot(tGps, errAver(2, :), 'r'), hold on
plot(tGps, sqrt(pAverDiag(2, :)), '--'), hold on
plot(tGps, -sqrt(pAverDiag(2, :)), '--')
title('Velocity Error', 'FontSize', 16)
xlabel('Times(s)', 'FontSize', 16)
ylabel('Velocity(m/s)', 'FontSize', 16);
subplot(3,1,3)
plot(tGps, errAver(3, :),'r'), hold on
plot(tGps, sqrt(pAverDiag(3, :)), '--'), hold on
plot(tGps, -sqrt(pAverDiag(3, :)), '--')
title('Bias Error', 'FontSize', 16)
xlabel('Times(s)', 'FontSize', 16)
ylabel('Bias', 'FontSize', 16);


figure(2)
subplot(3,3,1)
plot(tGps, reshape(pStoreAver(1, 1, :), 1, NOM) -...
    reshape(pAver(1, 1, :), 1, NOM))
title('P1, 1', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('Variance', 'FontSize', 12);
subplot(3,3,2)
plot(tGps, reshape(pStoreAver(1, 2, :), 1, NOM) -...
    reshape(pAver(1, 2, :), 1, NOM))
title('P1, 2', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
```

```
ylabel('Variance', 'FontSize', 12);
subplot(3,3,3)
plot(tGps, reshape(pStoreAver(1, 3, :), 1, NOM) -...
    reshape(pAver(1, 3, :), 1, NOM))
title('P1, 3', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('Variance', 'FontSize', 12);
subplot(3,3,4)
plot(tGps, reshape(pStoreAver(2, 1, :), 1, NOM) -...
    reshape(pAver(2, 1, :), 1, NOM))
title('P2, 1', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('Variance', 'FontSize', 12);
subplot(3,3,5)
plot(tGps, reshape(pStoreAver(2, 2, :), 1, NOM) -...
    reshape(pAver(2, 2, :), 1, NOM))
title('P2, 2', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('Variance', 'FontSize', 12);
subplot(3,3,6)
plot(tGps, reshape(pStoreAver(2, 3, :), 1, NOM) -...
    reshape(pAver(2, 3, :), 1, NOM))
title('P2, 3', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('Variance', 'FontSize', 12);
subplot(3,3,7)
plot(tGps, reshape(pStoreAver(3, 1, :), 1, NOM) -...
    reshape(pAver(3, 1, :), 1, NOM))
title('P3, 1', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('Variance', 'FontSize', 12);
subplot(3,3,8)
plot(tGps, reshape(pStoreAver(3, 2, :), 1, NOM) -...
    reshape(pAver(3, 2, :), 1, NOM))
title('P3, 2', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('Variance', 'FontSize', 12);
subplot(3,3,9)
plot(tGps, reshape(pStoreAver(3, 3, :), 1, NOM) -...
    reshape(pAver(3, 3, :), 1, NOM))
title('P3, 3', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('Variance', 'FontSize', 12);


figure(3)
subplot(3,3,1)
plot(tGps, reshape(orthEstErr(1, 1, :), 1, NOM))
title('E[.]1, 1', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('E[.]', 'FontSize', 12);
subplot(3,3,2)
plot(tGps, reshape(orthEstErr(1, 2, :), 1, NOM))
title('E[.]1, 2', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('E[.]', 'FontSize', 12);
```

```
subplot(3,3,3)
plot(tGps, reshape(orthEstErr(1, 3, :), 1, NOM))
title('E[.]1, 3', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('E[.]', 'FontSize', 12);
subplot(3,3,4)
plot(tGps, reshape(orthEstErr(2, 1, :), 1, NOM))
title('E[.]2, 1', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('E[.]', 'FontSize', 12);
subplot(3,3,5)
plot(tGps, reshape(orthEstErr(2, 2, :), 1, NOM))
title('E[.]2, 2', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('E[.]', 'FontSize', 12);
subplot(3,3,6)
plot(tGps, reshape(orthEstErr(2, 3, :), 1, NOM))
title('E[.]2, 3', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('E[.]', 'FontSize', 12);
subplot(3,3,7)
plot(tGps, reshape(orthEstErr(3, 1, :), 1, NOM))
title('E[.]3, 1', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('E[.]', 'FontSize', 12);
subplot(3,3,8)
plot(tGps, reshape(orthEstErr(3, 2, :), 1, NOM))
title('P3, 2', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('Variance', 'FontSize', 12);
subplot(3,3,9)
plot(tGps, reshape(orthEstErr(3, 3, :), 1, NOM))
title('E[.]3, 3', 'FontSize', 12)
xlabel('Times(s)', 'FontSize', 12)
ylabel('E[.]', 'FontSize', 12);
```