AE5335 Assignment 1

Submitted by: Purna Patel, Ritik Jain

Problem 3:

- Attitude Kinematics equations using Euler angles.

$$\begin{bmatrix}\dot{\psi}\\ \dot{\theta}\\ \dot{\phi}\end{bmatrix} = \frac{1}{C_\theta}\begin{bmatrix}0 & S_\phi & C_\phi\\ 0 & C_\phi C_\theta & -S_\phi C_\theta\\ C_\theta & S_\phi S_\theta & C_\phi S_\theta\end{bmatrix}\omega_{tb}^b$$

- Mechanization Equations.

$$\dot{V}^t = R_b^t a^b + g^t$$

- Considering constant velocity.

$$a^b = -R_b^t g^t = -g\begin{bmatrix}-S_\theta\\ S_\phi C_\theta\\ C_\phi C_\theta\end{bmatrix}$$

- Standard EKF Formulation.
  - State $x = (\psi, \theta, \phi)$
  - Input $u = r^b$ (Rate Gyro Measurements)
  - Measurement $z = \begin{bmatrix}m^b\\ a^b\end{bmatrix}$ (Compass magnetometer and accelerometer measurements)
  - Non-linear estimation model.

$$f(x,u,w) = \frac{1}{C_\theta}\begin{bmatrix}0 & S_\phi & C_\phi\\ 0 & C_\phi C_\theta & -S_\phi C_\theta\\ C_\theta & S_\phi S_\theta & C_\phi S_\theta\end{bmatrix}(u-w)$$

  - Non-linear measurement model.

$$h(x,u,\eta) = \begin{bmatrix}\psi\\ gS_\theta\\ -gS_\phi C_\theta\\ -gC_\phi C_\theta\end{bmatrix} + \eta$$

  - Process Noise Covariance: $Q = \sigma_r^2 I_{(3)}$
  - Measurement Noise Covariance: $R = \begin{bmatrix}\sigma_m^2 & 0_{(1\times3)}\\ 0_{(3\times1)} & \sigma_a^2 I_{(3)}\end{bmatrix}$
  - Linearization:

$$F = I_{(3)} + F_0 \Delta t$$

$$\text{Where, } F_0 = \begin{bmatrix} 0 & (qS_\phi + rC_\phi)T_\theta \sec\theta & (qC_\phi - rS_\phi)\sec\theta \\ 0 & 0 & -(qS_\phi + rC_\phi) \\ 0 & (qS_\phi + rC_\phi)\sec^2\theta & (qC_\phi - rS_\phi)T_\theta \end{bmatrix}$$

$$G_1 = H_0 \Delta t = -G_2$$

$$\text{Where, } H_0 = \frac{1}{C_\theta}\begin{bmatrix} 0 & S_\phi & C_\phi \\ 0 & C_\phi C_\theta & -S_\phi C_\theta \\ C_\theta & S_\phi S_\theta & C_\phi S_\theta \end{bmatrix}$$

$$C_{k-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & gC_\theta & 0 \\ 0 & gS_\phi S_\theta & -gC_\phi C_\theta \\ 0 & gC_\phi S_\theta & gS_\phi C_\theta \end{bmatrix}$$

- EFK Equations:
  - Estimation using non-linear process model.

$$\hat{x}^- = \hat{x}_{k-1} + f(\hat{x}_{k-1}, u_{k-1}, w_{k-1})\Delta t$$
$$P^- = F_{k-1}P_{k-1}F_{k-1}^T + G_{2_{k-1}}QG_{2_{k-1}}^T$$

  - Calculating Kalman Gain.

$$L_k = P^-C^T(CP^-C^T + R)^{-1}$$

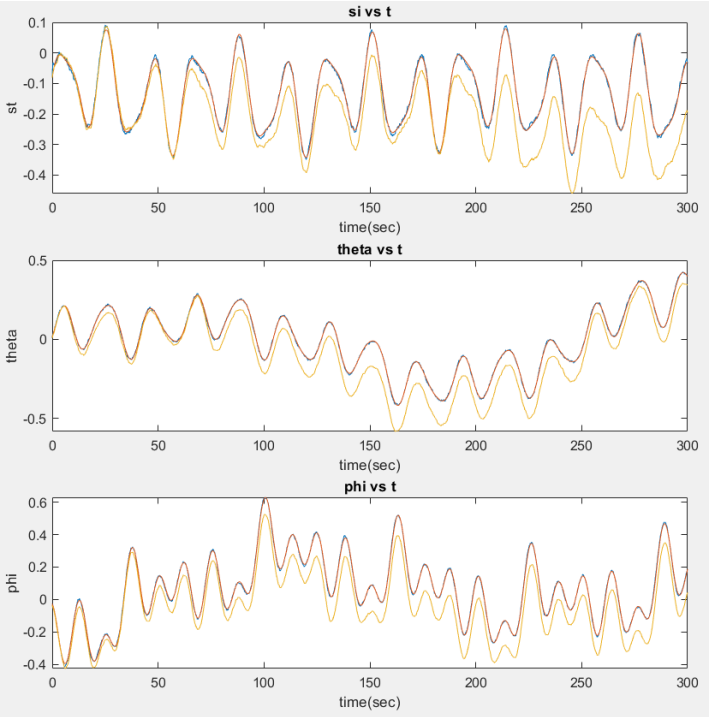
  - Updating estimates.

$$\hat{x}_k = \hat{x}^- + L_k(z_k - h(\hat{x}^-))$$

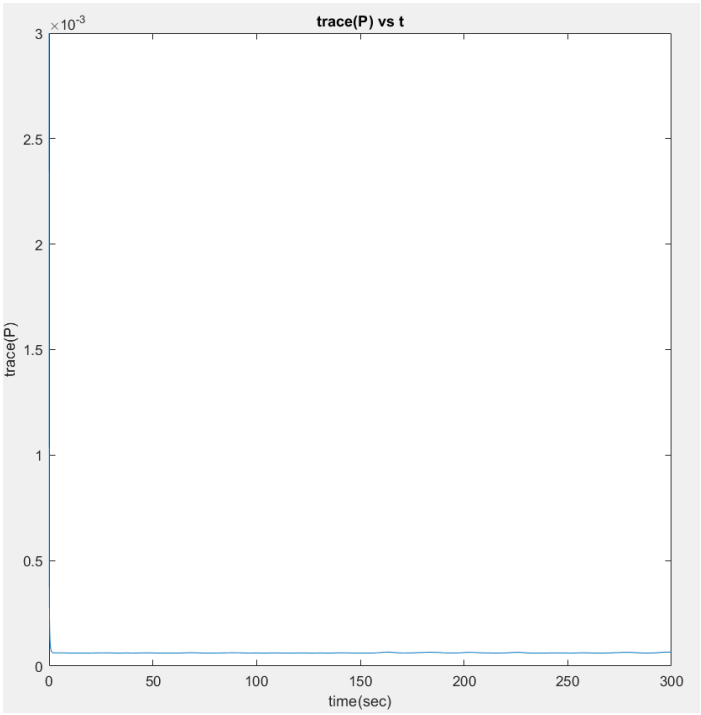
  - Updating estimation Covariance.

$$P_k = (I_{(3)} - L_kC)P^-$$

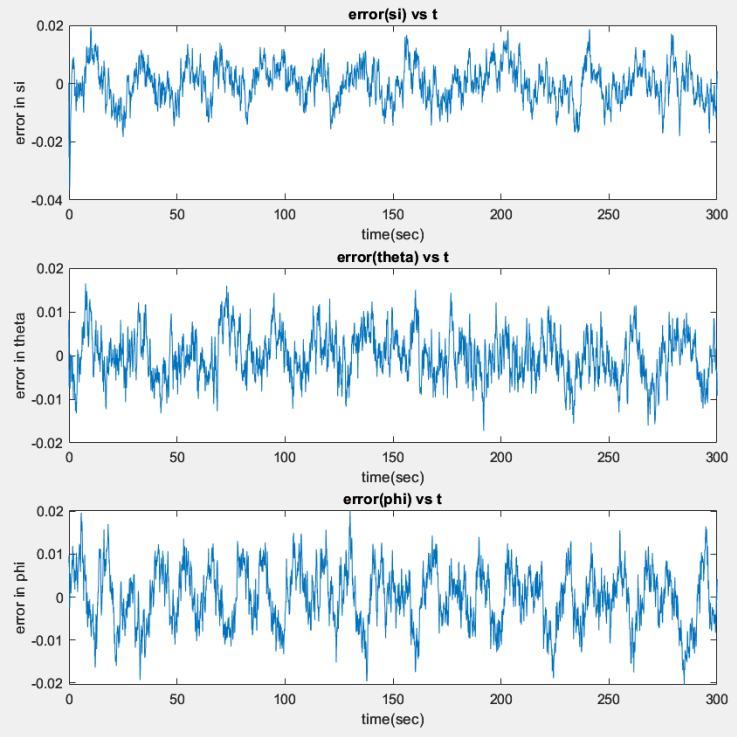
- **Results:**



- In the above plot, the red curve represents the actual Euler angles, the blue curve represents the EKF estimates, and the yellow curve represents the estimates only based on process model integration.
- It can be observed that the EKF estimates coincides with the actual Euler angles.
- Though there are some jitters observed in the estimates due to noisy readings.



- The above plot shows the propagation of the trace of the covariance matrix P.
- It quickly decreases to a value close to zero.



- The above plot displays the propagation of errors between estimated states and actual states.
- It can be observed that the error oscillates about a mean of zero.
- It can be observed that despite of the starting point, the errors settle to mean of zero.

MATLAB Script attached at the end of the report.

Problem 4:

- Standard EKF Formulation with sensor biases.
  - State $\boldsymbol{x} = (\psi, \theta, \phi, \boldsymbol{b}_a, \boldsymbol{b}_r, b_m)$
  - Input $\boldsymbol{u} = \boldsymbol{r}^b$ (Rate Gyro Measurements)
  - Measurement $\boldsymbol{z} = \begin{bmatrix} m^b \\ \boldsymbol{a}^b \end{bmatrix}$ (Compass magnetometer and accelerometer measurements)
  - Non-linear estimation model.

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) = \begin{bmatrix} \frac{1}{C_\theta} \begin{bmatrix} 0 & S_\phi & C_\phi \\ 0 & C_\phi C_\theta & -S_\phi C_\theta \\ C_\theta & S_\phi S_\theta & C_\phi S_\theta \end{bmatrix} (\boldsymbol{u} - \boldsymbol{b}_r - \boldsymbol{w}) \\ 0_{(7\times 1)} \end{bmatrix}$$

  - Non-linear measurement model.

$$\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\eta}) = \begin{bmatrix} \psi \\ gS_\theta \\ -gS_\phi C_\theta \\ -gC_\phi C_\theta \end{bmatrix} + \begin{bmatrix} b_m \\ \boldsymbol{b}_a \end{bmatrix} + \boldsymbol{\eta}$$

  - Process Noise Covariance: $Q = \sigma_r^2 I_{(3)}$
  - Measurement Noise Covariance: $R = \begin{bmatrix} \sigma_m^2 & 0_{(1\times 3)} \\ 0_{(3\times 1)} & \sigma_a^2 I_{(3)} \end{bmatrix}$
  - Linearization:

$$F_0 = \begin{bmatrix} 0 & (qS_\phi + rC_\phi)T_\theta \sec\theta & (qC_\phi - rS_\phi)\sec\theta \\ 0 & 0 & -(qS_\phi + rC_\phi) \\ 0 & (qS_\phi + rC_\phi)\sec^2\theta & (qC_\phi - rS_\phi)T_\theta \end{bmatrix}$$

$$H_0 = \frac{1}{C_\theta} \begin{bmatrix} 0 & S_\phi & C_\phi \\ 0 & C_\phi C_\theta & -S_\phi C_\theta \\ C_\theta & S_\phi S_\theta & C_\phi S_\theta \end{bmatrix}$$

$$F = I_{(10)} + \begin{bmatrix} F_0 & 0_{(3)} & -H_0 & 0_{(3\times 1)} \\ & & 0_{(7\times 10)} & \end{bmatrix} \Delta t$$

$$G_1 = \begin{bmatrix} H_0 \\ 0_{(7\times 3)} \end{bmatrix} \Delta t = -G_2$$

$$C_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & gC_\theta & 0 \\ 0 & gS_\phi S_\theta & -gC_\phi C_\theta \\ 0 & gC_\phi S_\theta & gS_\phi C_\theta \end{bmatrix}$$

$$C_{k-1} = \begin{bmatrix} C_0 & 0_{(1\times6)} & 1 \\ & I_{(3)} & 0_{(3\times4)} \end{bmatrix}$$

- EFK Equations:
  - Estimation using non-linear process model.

$$\hat{x}^- = \hat{x}_{k-1} + f(\hat{x}_{k-1}, u_{k-1}, w_{k-1})\Delta t$$
$$P^- = F_{k-1}P_{k-1}F_{k-1}^T + G_{2_{k-1}}QG_{2_{k-1}}^T$$

  - Calculating Kalman Gain.

$$L_k = P^-C^T(CP^-C^T + R)^{-1}$$

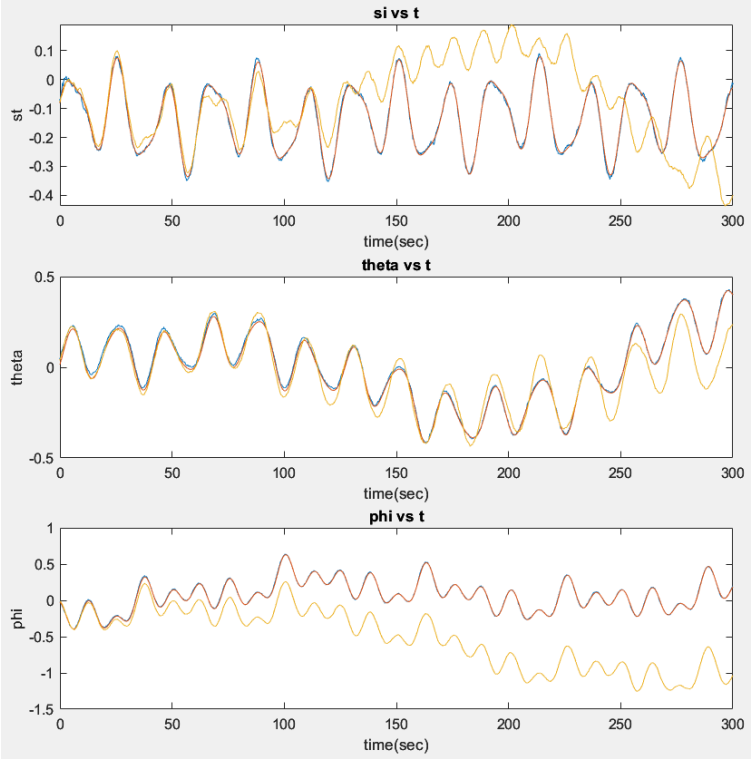
  - Updating estimates.

$$\hat{x}_k = \hat{x}^- + L_k(z_k - h(\hat{x}^-))$$

  - Updating estimation Covariance.

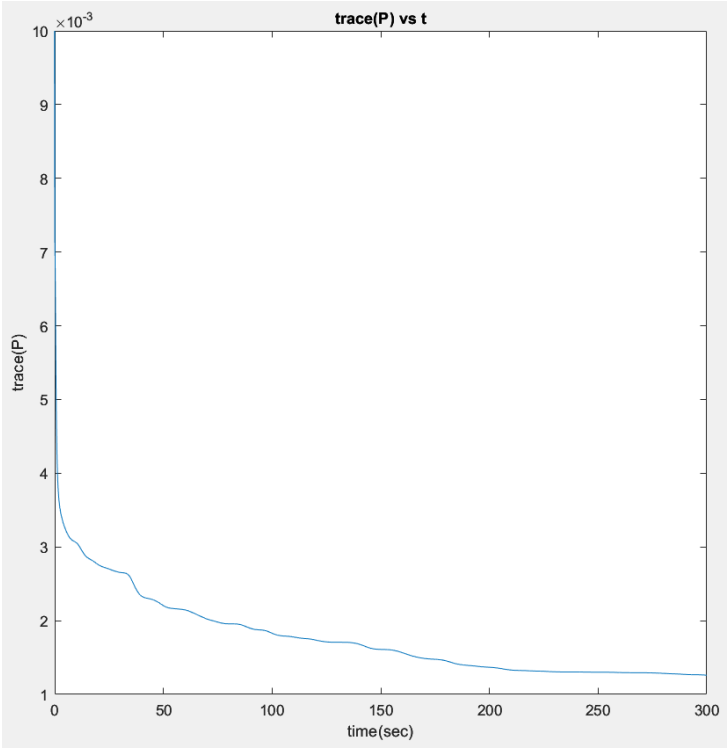
$$P_k = \left(I_{(3)} - L_kC\right)P^-$$

- **Result:**

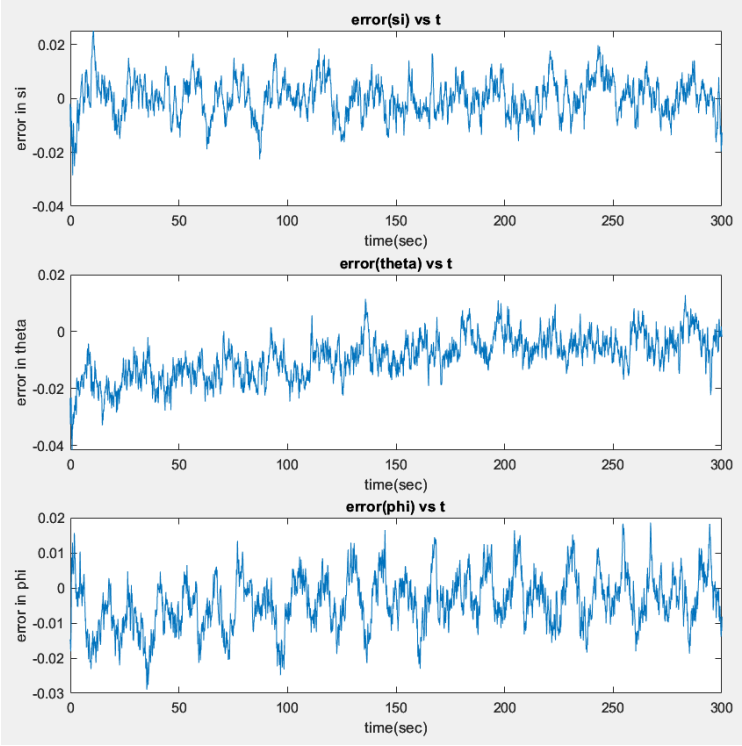


- In the above plot, the red curve represents the actual Euler angles, the blue curve represents the EKF estimates, and the yellow curve represents the estimates only based on process model integration.

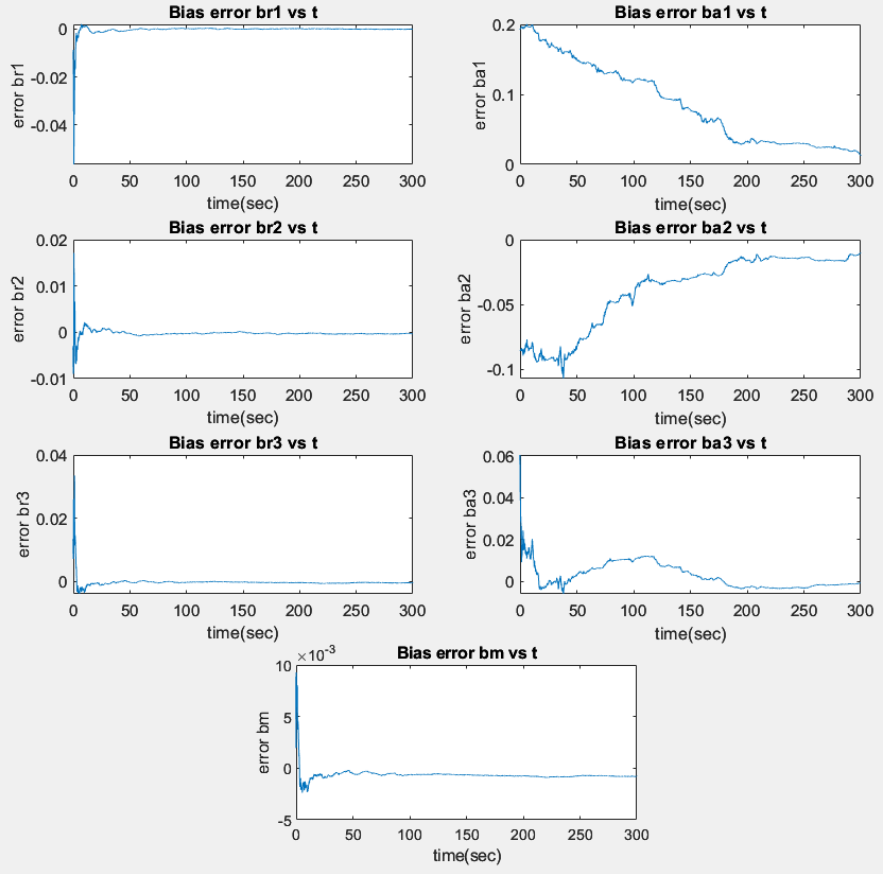
- It can be observed that the EKF estimates coincides with the actual Euler angles while the only-integral estimates (yellow curve) varies too much due to sensor bias.
- Though there are some jitters observed in the estimates due to noisy readings.



- The above plot shows the propagation of the trace of the covariance matrix P.
- It can be observed that it decreases to a value close to zero.



- The above plot displays the propagation of errors between estimated states and actual states.
- It can be observed that the error oscillates about a mean of zero.
- It can be observed that despite of the starting point, the errors settle to mean of zero.



- The above plots display the propagation of errors between the actual sensor biases and the estimated sensor biases.
- It can be observed that the errors settle close to zero as the time proceeds.

MATLAB Script attached at the end of the report.

Problem 5:

- Quaternion Kinematics:

$$\begin{bmatrix}\dot{e}_0\\ \dot{e}_1\\ \dot{e}_2\\ \dot{e}_3\end{bmatrix}=\frac{1}{2}\begin{bmatrix}0 & -p & -q & -r\\ p & 0 & r & -q\\ q & -r & 0 & p\\ r & q & -p & 0\end{bmatrix}\begin{bmatrix}e_0\\ e_1\\ e_2\\ e_3\end{bmatrix}=\frac{1}{2}\begin{bmatrix}-e_1 & -e_2 & -e_3\\ e_0 & -e_3 & e_2\\ e_3 & e_0 & -e_1\\ -e_2 & e_1 & e_0\end{bmatrix}\boldsymbol{\omega}_{tb}^{b}$$

- Mechanization Equations.

$$\dot{V}^t = R_b^t a^b + g^t$$

- Considering constant velocity.

$$a^b = -R_b^t g^t = -g^b$$

$$\begin{bmatrix}0\\ g^b\end{bmatrix} = e^* \otimes \left(\begin{bmatrix}0\\ g^t\end{bmatrix} \otimes e\right)$$

$$\begin{bmatrix}0\\ g^b\end{bmatrix} = \begin{bmatrix}e_0\\ -e_1\\ -e_2\\ -e_3\end{bmatrix} \otimes \left(\begin{bmatrix}0\\ 0\\ 0\\ g\end{bmatrix} \otimes \begin{bmatrix}e_0\\ e_1\\ e_2\\ e_3\end{bmatrix}\right)$$

$$\begin{bmatrix}0\\ g^b\end{bmatrix} = \begin{bmatrix}e_0\\ -e_1\\ -e_2\\ -e_3\end{bmatrix} \otimes \begin{bmatrix}0-0-0-ge_3\\ 0+0+0-ge_2\\ 0-0+0+ge_1\\ 0+0-0+ge_0\end{bmatrix} = \begin{bmatrix}e_0\\ -e_1\\ -e_2\\ -e_3\end{bmatrix} \otimes \begin{bmatrix}-ge_3\\ -ge_2\\ ge_1\\ ge_0\end{bmatrix}$$

$$\begin{bmatrix}0\\ g^b\end{bmatrix} = g\begin{bmatrix}-e_0e_3 - e_1e_2 + e_2e_1 + e_0e_3\\ -e_0e_2 + e_1e_3 - e_2e_0 + e_1e_3\\ e_0e_1 + e_1e_0 + e_2e_3 + e_3e_2\\ e_0^2 - e_1^2 - e_2^2 + e_3^2\end{bmatrix}$$

$$\begin{bmatrix}0\\ g^b\end{bmatrix} = g\begin{bmatrix}0\\ 2(-e_0e_2 + e_1e_3)\\ 2(e_0e_1 + e_2e_3)\\ e_0^2 - e_1^2 - e_2^2 + e_3^2\end{bmatrix}$$

$$a^b = -g\begin{bmatrix}2(-e_0e_2 + e_1e_3)\\ 2(e_0e_1 + e_2e_3)\\ e_0^2 - e_1^2 - e_2^2 + e_3^2\end{bmatrix}$$

- Mechanization equation for magnetometer reading.

$$m^b = \tan^{-1}\left(\frac{2(e_0e_3 + e_1e_2)}{e_0^2 + e_1^2 - e_2^2 - e_3^2}\right)$$

- Standard EKF Formulation with sensor biases.
  - State $\boldsymbol{x} = (e_0, e_1, e_2, e_3, \boldsymbol{b}_a, \boldsymbol{b}_r, b_m)$
  - Input $\boldsymbol{u} = \boldsymbol{r}^b$ (Rate Gyro Measurements)
  - Measurement $\boldsymbol{z} = \begin{bmatrix} m^b \\ \boldsymbol{a}^b \end{bmatrix}$ (Compass magnetometer and accelerometer measurements)
  - Linear estimation model.

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) = \begin{bmatrix} \frac{1}{2}\begin{bmatrix} -e_1 & -e_2 & -e_3 \\ e_0 & -e_3 & e_2 \\ e_3 & e_0 & -e_1 \\ -e_2 & e_1 & e_0 \end{bmatrix}(\boldsymbol{u} - \boldsymbol{b}_r - \boldsymbol{w}) \\ 0_{(7\times1)} \end{bmatrix}$$

  - Non-linear measurement model.

$$\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\eta}) = \begin{bmatrix} \tan^{-1}\left(\frac{2(e_0e_3 + e_1e_2)}{e_0^2 + e_1^2 - e_2^2 - e_3^2}\right) \\ -2g(-e_0e_2 + e_1e_3) \\ -2g(e_0e_1 + e_2e_3) \\ -g(e_0^2 - e_1^2 - e_2^2 + e_3^2) \end{bmatrix} + \begin{bmatrix} b_m \\ \boldsymbol{b}_a \end{bmatrix} + \boldsymbol{\eta}$$

  - Process Noise Covariance: $Q = \sigma_r^2 I_{(3)}$
  - Measurement Noise Covariance: $R = \begin{bmatrix} \sigma_m^2 & 0_{(1\times3)} \\ 0_{(3\times1)} & \sigma_a^2 I_{(3)} \end{bmatrix}$
  - Linearization:

$$F_0 = \frac{1}{2}\begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix}$$

$$H_0 = \frac{1}{2}\begin{bmatrix} -e_1 & -e_2 & -e_3 \\ e_0 & -e_3 & e_2 \\ e_3 & e_0 & -e_1 \\ -e_2 & e_1 & e_0 \end{bmatrix}$$

$$F = I_{(11)} + \begin{bmatrix} F_0 & 0_{(4\times3)} & -H_0 & 0_{(4\times1)} \\ & 0_{(7\times11)} & & \end{bmatrix}\Delta t$$

$$G_1 = \begin{bmatrix} H_0 \\ 0_{(7\times3)} \end{bmatrix}\Delta t = -G_2$$

  - Linearization of measurement model:
    Let $Y = 2(e_0e_3 + e_1e_2)$ and $X = e_0^2 + e_1^2 - e_2^2 - e_3^2$

$$\psi = \tan^{-1}\left(\frac{Y}{X}\right)$$

$$\frac{\partial \psi}{\partial e_0} = \frac{1}{1+\frac{Y^2}{X^2}}\left(\frac{X(2e_3) - Y(2e_0)}{X^2}\right) = \frac{2e_3X - 2e_0Y}{X^2+Y^2}$$

$$\frac{\partial \psi}{\partial e_1} = \frac{2e_2X - 2e_1Y}{X^2+Y^2}$$

$$\frac{\partial \psi}{\partial e_2} = \frac{2e_1X + 2e_2Y}{X^2+Y^2}$$

$$\frac{\partial \psi}{\partial e_3} = \frac{2e_0X + 2e_3Y}{X^2+Y^2}$$

$$C_0 = \begin{bmatrix} \frac{\partial \psi}{\partial e_0} & \frac{\partial \psi}{\partial e_1} & \frac{\partial \psi}{\partial e_2} & \frac{\partial \psi}{\partial e_3} \\ 2ge_2 & -2ge_3 & 2ge_0 & -2ge_1 \\ -2ge_1 & -2ge_0 & -2ge_3 & -2ge_2 \\ -2ge_0 & 2ge_1 & 2ge_2 & -2ge_3 \end{bmatrix}$$

$$C_{k-1} = \begin{bmatrix} C_0 & 0_{(1\times 6)} & 1 \\ & I_{(3)} & 0_{(3\times 4)} \end{bmatrix}$$

- Updated Measurement Model:
  - Better performance was observed by rearranging the measurement model as follows using unit quaternion property.

$$\boldsymbol{h}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{\eta}) = \begin{bmatrix} \tan^{-1}\left(\frac{2(e_0e_3 + e_1e_2)}{e_0^2 + e_1^2 - e_2^2 - e_3^2}\right) \\ -2g(-e_0e_2 + e_1e_3) \\ -2g(e_0e_1 + e_2e_3) \\ -g(1 - 2(e_1^2 + e_2^2)) \end{bmatrix} + \begin{bmatrix} b_m \\ \boldsymbol{b}_a \end{bmatrix} + \boldsymbol{\eta}$$

  - Updated Linearized Measurement model.

$$C_0 = \begin{bmatrix} \frac{\partial \psi}{\partial e_0} & \frac{\partial \psi}{\partial e_1} & \frac{\partial \psi}{\partial e_2} & \frac{\partial \psi}{\partial e_3} \\ 2ge_2 & -2ge_3 & 2ge_0 & -2ge_1 \\ -2ge_1 & -2ge_0 & -2ge_3 & -2ge_2 \\ 0 & 4ge_1 & 4ge_2 & 0 \end{bmatrix}$$

- EFK Equations:
  - Estimation using non-linear process model.

$$\hat{x}^- = \hat{x}_{k-1} + f(\hat{x}_{k-1}, u_{k-1}, w_{k-1})\Delta t$$
$$P^- = F_{k-1}P_{k-1}F_{k-1}^T + G_{2_{k-1}}QG_{2_{k-1}}^T$$

  - Calculating Kalman Gain.

$$L_k = P^-C^T(CP^-C^T + R)^{-1}$$

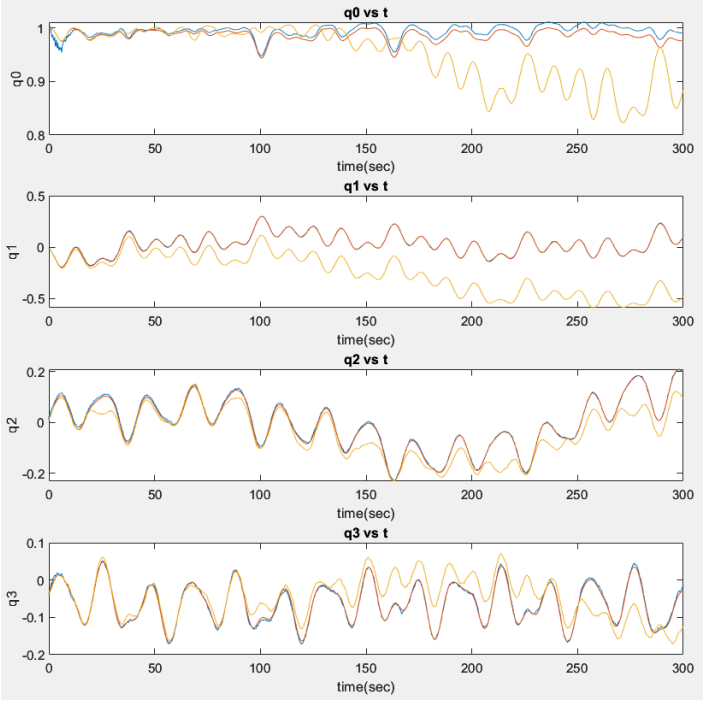
  - Updating estimates.

$$\hat{x}_k = \hat{x}^- + L_k(z_k - h(\hat{x}^-))$$

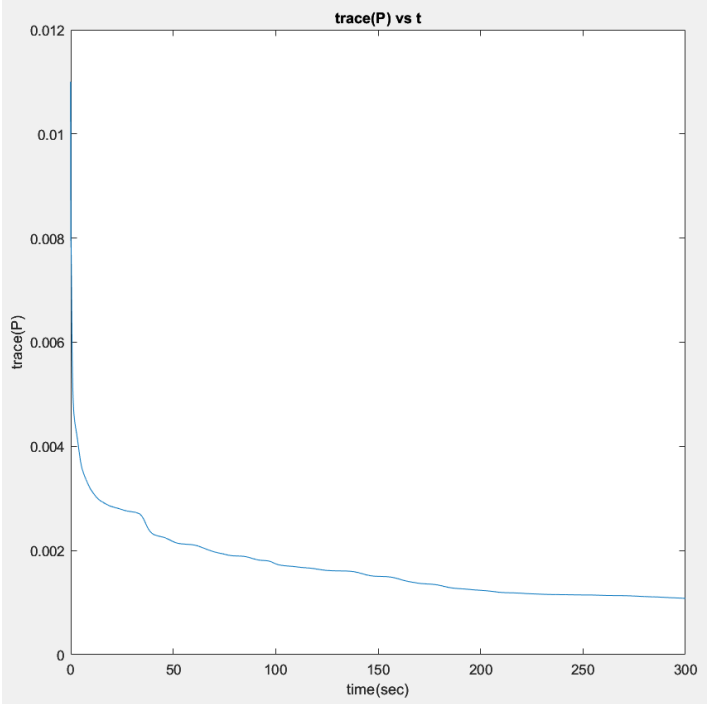
  - Updating estimation Covariance.

$$P_k = \left(I_{(3)} - L_kC\right)P^-$$

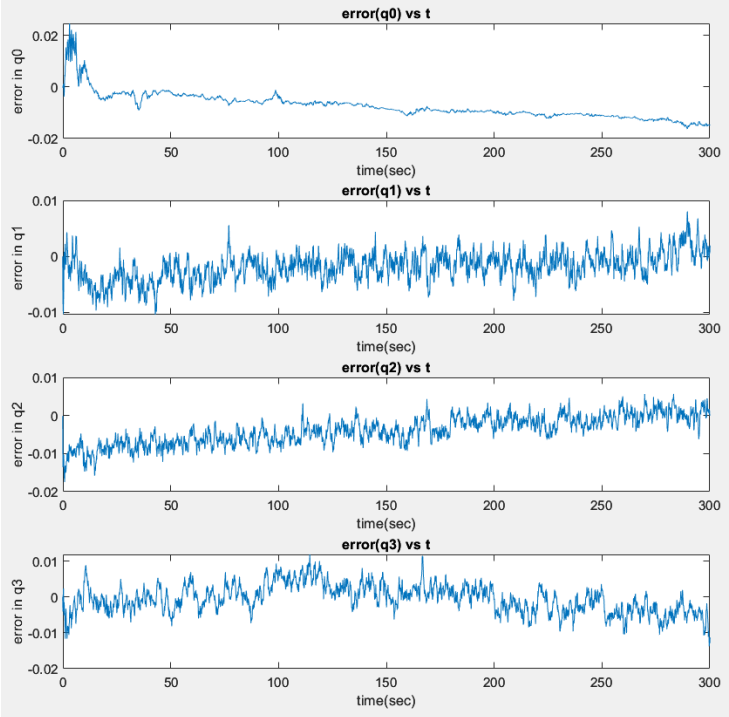
- **Results:**



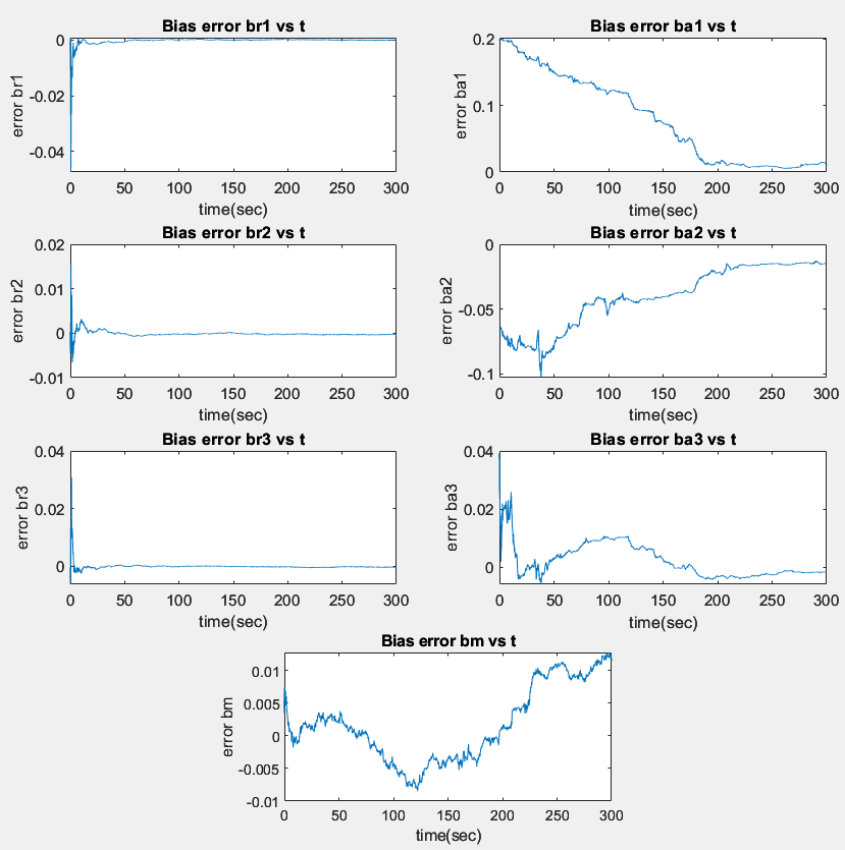
- In the above plot, the red curve represents the actual Quaternions, the blue curve represents the EKF estimates, and the yellow curve represents the estimates only based on process model integration.
- It can be observed that the EKF estimates coincides with the actual quaternions for q1, q2 and q3 while there is some error observed in q0.
- The only-integral estimates (yellow curve) varies too much due to sensor bias.
- There are some jitters observed in the estimates due to noisy readings.



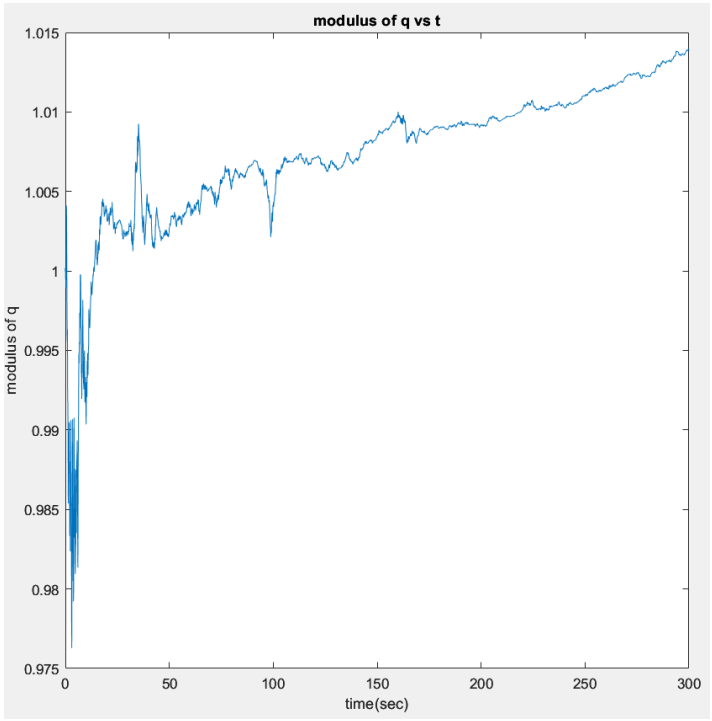
- The above plot shows the propagation of the trace of the covariance matrix P.
- It can be observed that it decreases to a value close to zero.



- The above plot displays the propagation of errors between estimated states and actual states.
- It can be observed that the error in q0 increases slightly over time.
- Errors in other 3 states settle to a mean of 0 as the time proceeds.



- The above plots display the propagation of errors between the actual sensor biases and the estimated sensor biases.
- It can be observed that the errors in rate gyro sensor bias quickly settle close to zero.
- While other sensor biases reach zero after certain time.



- The above plot shows the modulus of quaternions estimated to check if it follows the condition for unit quaternion.
- It can be seen that the modulus of quaternions always remains around 1.

MATLAB Script attached at the end of the report

Problem 6:

The paper proposes a solution using a monocular camera and Inertial sensor for the problem of navigation and control of Unmanned Aircraft with no GPS system. This problem is proposed due the fact that GPS sensors emits signals which can be traced which might be undesirable in some conditions. Also, while using an aerial vehicle indoor there can be situation the GPS signals would be weak. The use of a premade map to navigate is not always the optimal solution as it will fail if the vehicle enters an unmapped area, this creates a bottleneck in the system. Using periodic global map optimization and localization from the resulting overwhelms computational resources and results in a slow drift in attitude, which can be a concern for long duration flight.

The reason of using a monocular camera is that it will keep the algorithm less computationally intensive. Using a stereo camera can make the system computationally intensive which causes other feature of the vehicle to compromise. As aerial vehicles can't have heavy processors like land vehicles due to weight constraints using an efficient algorithm becomes important. Similarly SLAM techniques in real time is difficult due the fact aerial vehicles have fast dynamics we need to have a fast closed loop control.

The paper focuses on three major aspects, having an algorithm that can perform the closed loop control with the fast dynamics. They focus on two major sequences, one navigating from point A to B and two, the landing sequence which are the crucial factors for any aerial vehicle.

They use an online feature database to match features to get frame to frame transition. These features are extracted from the monocular camera using Harris conners and Mahalanobis distance is used to match features from frame-to-frame. This database is updated using relative confidence of each feature. A new feature is added to the database if it is not already there in the system with an initial confidence score which depends on the number of features being added in that moment. If there are multiple features added the initial confidence is set low, and when few features are added the confidence is high. If the features are repeated the confidence increases and similarly if the feature does not appear the confidence drops. If the confidence drops the initial confidence threshold it is removed from the database. This keeps the database updated to remove unnecessary feature matching and removes unwanted feature quickly.

The Inertial sensors are used to predict the frame-to-frame transition. The inertial sensor used in the paper are the accelerometers and gyroscopes of the IMU to measure accelerations and angular rates. To enable closed-loop feedback control for autonomous operation, UAVs need an accurate estimate of their position and attitude, for which they use Extended Kalman Filter. It is necessary to estimate vehicle's position, velocity, quaternion attitude, IMU acceleration and gyroscope biases.

To test their performance they use two vehicles, medium-sized UAV about 66 kg to operate in unmapped outdoor environments, and the second one is a small-sized (Micro) M-UAV of 1 kg designed to operate in cluttered and confined unmapped indoor and outdoor environments. The result shows that a maximum of 50 features were stored in the database and over the 560 m flight the final navigation error was 5 m. This performed better than most existing research available, except there but those algorithms are only limited to 8 minutes of flight time,

whereas this paper have presented over double that time. The method overcomes the shortcomings of existing V-INS algorithms, such as computational inefficiency and gives proven flight-test results for both indoors and outdoors navigation.

Problem 7:

This paper also aims to solve the problem of a GPS denied navigation for aerial vehicles. The paper proposes the use of Inertial sensor and a wide-field optical flow information. They use Unscented Information Filter to estimate the ground velocity and attitude states. They use two different formulations, first a full state formulation including velocity and attitude and second a simplified formulation which assumes that the lateral and vertical velocity of the aircraft are negligible. In addition, measurements of a laser rangefinder sensor were used to recover the image distance. The idea is to enable navigation in urban areas where GPS is denied, and GPS sensors are not self-constrained.

The paper is inspired by honeybees that demonstrated impressive capabilities in flight navigation without receiving external communications. Using a Vision data correctly with other onboard sensors measurements to get a stable navigation system. For which feature detection algorithm they use Scale-Invariant Feature Transform (SIFT) which helps to get optical flow vectors. Benefits of optical flow is that it implicitly contains information about the aircraft attitude angles. The technique does not rely on horizon detection as it may not be visible all the time. Using a wide-field optical flow also gives us roll pitch information that is not observed from image center.

The estimation of ground velocity and Euler attitude angle is done by fussing Inertial Measurement of bode axis acceleration and angular rates along with the optical flow measurements and laser rangefinder measurements. For the full formulation a first order Gauss-Markov noise model was used to define the dynamics for the bias parameters. For this paper they assume each optical flow measurement carries equal uncertainty and are uncorrelated. For the simplified formulation the assumption is made that v and w are zero which removes the acceleration in y and z axis and only the x axis optical flow is relevant. This reduces a large number of updates and hence reduce the computation for optical flow measurements and achieves a higher sampling rate. Lastly it is possible to include range finder in the state estimation with the assumption that the ground is flat. Finally, the use of the Unscented Information Filter (UIF) over Kalman filter is that redundant information vectors are additive therefore, the time-varying number of outputs obtained from optical flow can easily be handled with relatively lower computation.

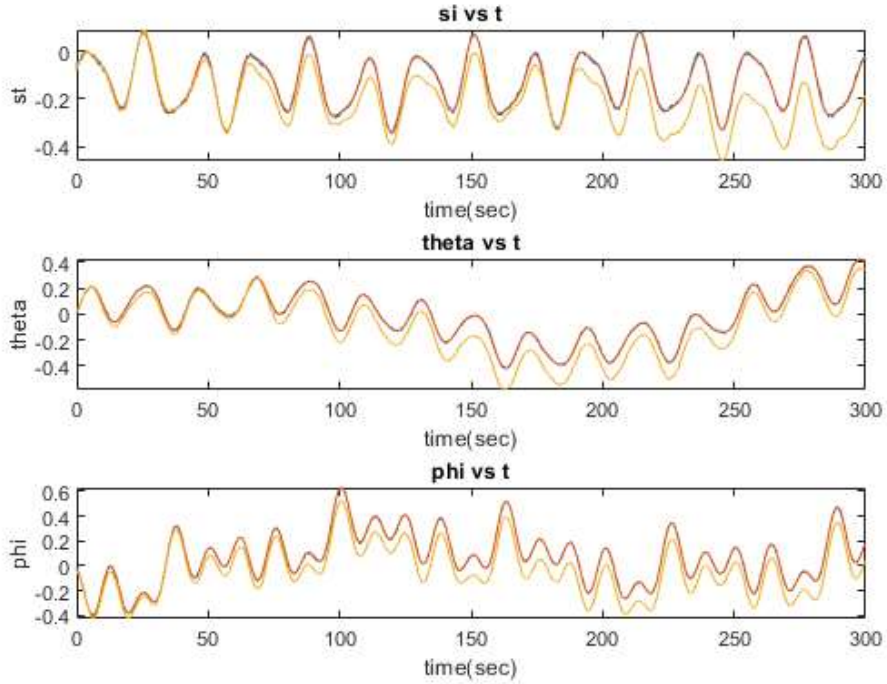
To test the results, they collected data from a UAV which had a GPS sensor which served as ground truth, along with it had a camera collecting video feeds. Two different datasets were created to test the approach. No real time testing of the algorithm was done. The results for the full formulation we see that the error for forward velocity is reasonable low than lateral velocity. This because observability issues with optical flow for lateral velocity. For the simplified formulation the mean error significantly increased due to the assumption of lateral velocity being zero and hence the performance decreased. Finally, after adding the laser rangefinder the error improved for the full formulation and the simplified formulation was not affected significantly. The paper concludes that both the formulations are effective in regulating the inertial drift and the having a rangefinder with the full formulation shows effective in estimating ground velocity and attitude angles for both test data.

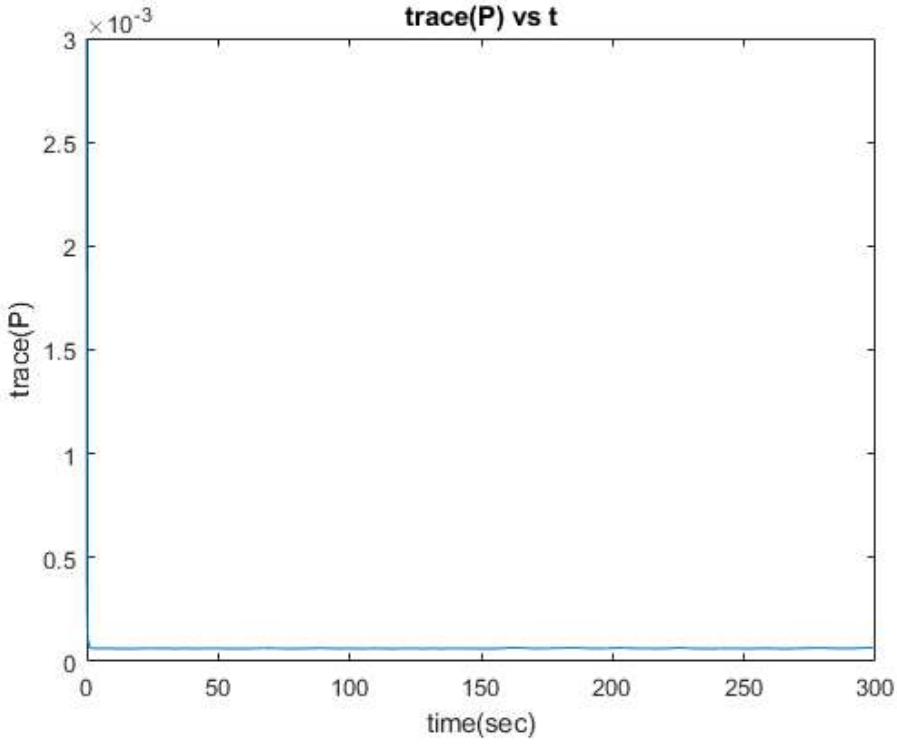
Problem 3 MATLAB Script

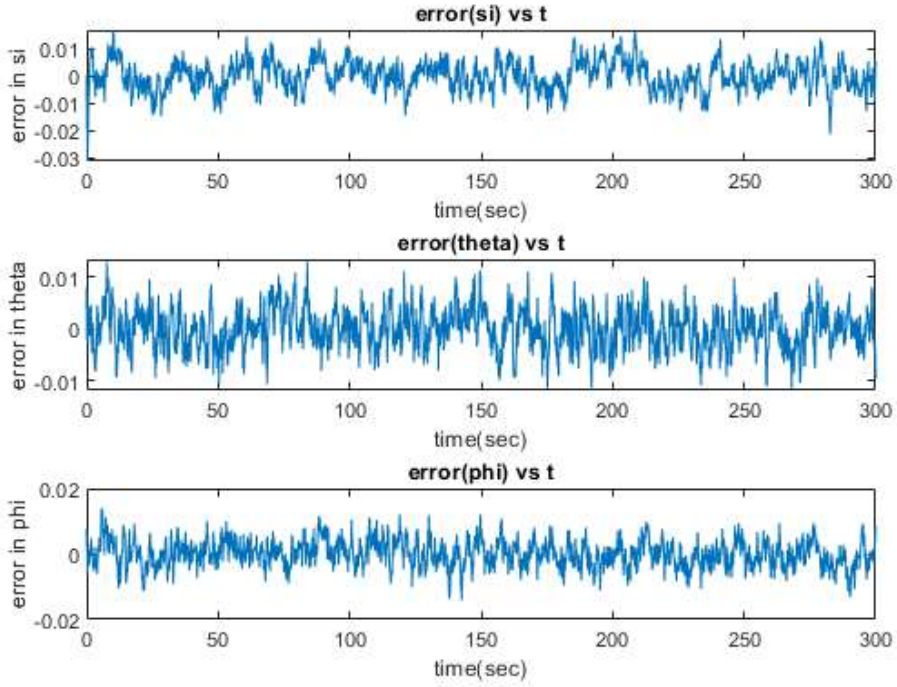
```
clear;
clc;

load ae5335_asg1_prob1_synthetic_data.mat

n_pts = numel(timestamps);
g = 9.81;

compass_cov = std_dev_compass^2;
accel_cov = (std_dev_accel^2)*eye(3);
R_cov = [compass_cov zeros(1,3); zeros(3,1) accel_cov];                    %Defining measurement error covariance
Q_cov = (std_dev_rategyro^2)*eye(3);                                       %Defining process error covariance
P_cov = eye(3)*0.001;                                                     %Initilizing estimation error covariance

%Initlizing states
si_initial = z_compass_mb(1);
theta_initial = asin(z_accelerometer_ab(1,1)/9.81);
tan_phi = z_accelerometer_ab(2,1)/z_accelerometer_ab(3,1);
phi_initial = atan(tan_phi);
x_hat = [si_initial; theta_initial; phi_initial];
x_int = x_hat;


%Initilizing Data matrix for plots
x_hat_plot = [];
x_int_plot = [];
P_trace_plot = [];

%Saving First data set
x_hat_plot(:,1) = x_hat;
x_int_plot(:,1) = x_int;
P_trace_plot(:,1) = trace(P_cov);

for k = 2 : n_pts

    dt = timestamps(k) - timestamps(k-1);

    si = x_hat(1);
    theta = x_hat(2);
    phi = x_hat(3);

    % Rate Gyro readings
    u_k1 = z_rategyro_rb(:,k-1);

    H0 = [0           sin(phi)              cos(phi);
          0           cos(phi)*cos(theta)   -sin(phi)*cos(theta);
          cos(theta)  sin(phi)*sin(theta)   cos(phi)*sin(theta)];
    H0 = (1/cos(theta))*H0;

    %Pridicting Estimates using nonlinear model
    x_hat_dot = H0*u_k1;
    x_hat_minus = x_hat + x_hat_dot * dt;

    %Linearization
    F0 = [...
        0   (u_k1(2)*sin(phi)+u_k1(3)*cos(phi))*tan(theta)*sec(theta)  (u_k1(2)*cos(phi)-u_k1(3)*sin(phi))*sec(theta);
        0   0                                                          -(u_k1(2)*sin(phi)+u_k1(3)*cos(phi));
        0   (u_k1(2)*sin(phi)+u_k1(3)*cos(phi))*(sec(theta))^2         (u_k1(2)*cos(phi)-u_k1(3)*sin(phi))*tan(theta)];

    F_k1 = eye(3) + F0*dt;
    G2_k1 = -H0*dt;
```

```
    P_minus = F_k1*P_cov*F_k1' + G2_k1*Q_cov*G2_k1';

    %Linearization of Measurement Model
    C_k1 = [1   0                       0;
            0   g*cos(theta)            0;
            0   g*sin(phi)*sin(theta)  -g*cos(phi)*cos(theta);
            0   g*cos(phi)*sin(theta)   g*sin(phi)*cos(theta)];

    %Calculating Kalman Gain
    L_k = P_minus*C_k1'/(C_k1*P_minus*C_k1' + R_cov);

    %Aquiring Compass and Acclelrometer readings
    z_k = [z_compass_mb(:,k);z_accelerometer_ab(:,k)];

    si = x_hat_minus(1);
    theta = x_hat_minus(2);
    phi = x_hat_minus(3);
    h_x_hat_minus = [si; g*sin(theta); -g*sin(phi)*cos(theta); -g*cos(phi)*cos(theta)];

    %Updating Estimetes
    x_hat = x_hat_minus + L_k*(z_k - h_x_hat_minus);
    P_cov = (eye(3) - L_k*C_k1)*P_minus;


    %Calculating only pridictons
    H0_int = [0             sin(x_int(3))                   cos(x_int(3));
          0             cos(x_int(3))*cos(x_int(2))     -sin(x_int(3))*cos(x_int(2));
          cos(x_int(2)) sin(x_int(3))*sin(x_int(2))     cos(x_int(3))*sin(x_int(2))];
    H0_int = (1/cos(x_int(2)))*H0_int;

    x_int_dot = H0_int*u_k1;
    x_int = x_int + x_int_dot * dt;

    x_hat_plot(:,k) = x_hat;
    x_int_plot(:,k) = x_int;
    P_trace_plot(:,k) = trace(P_cov);

end

figure;
% plot 1
subplot(3,1,1)
plot(timestamps,x_hat_plot(1,:))
title('si vs t')
xlabel 'time(sec)';
ylabel 'st';
hold on;
plot(timestamps,euler_angles_true(1,:));
plot (timestamps,x_int_plot(1,:))

% plot 2
subplot(3,1,2)
plot(timestamps,x_hat_plot(2,:))
title('theta vs t')
xlabel 'time(sec)';
ylabel 'theta';
hold on;
plot(timestamps,euler_angles_true(2,:));
plot (timestamps,x_int_plot(2,:))

% plot 3
subplot(3,1,3)
plot(timestamps,x_hat_plot(3,:))
title('phi vs t')
```

```
xlabel 'time(sec)';
ylabel 'phi';
hold on;
plot(timestamps,euler_angles_true(3,:));
plot (timestamps,x_int_plot(3,:))

figure;
plot(timestamps,P_trace_plot)
title('trace(P) vs t')
xlabel 'time(sec)';
ylabel 'trace(P)';

figure;
% plot 1
subplot(3,1,1)
plot(timestamps,(euler_angles_true(1,:)-x_hat_plot(1,:)))
title('error(si) vs t')
xlabel 'time(sec)';
ylabel 'error in si';

% plot 2
subplot(3,1,2)
plot(timestamps,(euler_angles_true(2,:)-x_hat_plot(2,:)))
title('error(theta) vs t')
xlabel 'time(sec)';
ylabel 'error in theta';

% plot 3
subplot(3,1,3)
plot(timestamps,(euler_angles_true(3,:)-x_hat_plot(3,:)))
title('error(phi) vs t')
xlabel 'time(sec)';
ylabel 'error in phi';
```


si vs t
st
time(sec)
theta vs t
theta
time(sec)
phi vs t
phi
time(sec)


trace(P) vs t
×10^-3
trace(P)
time(sec)



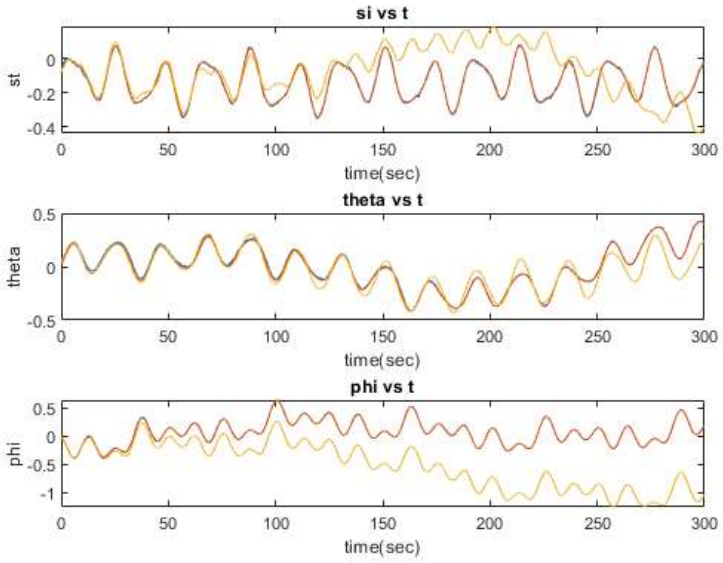
Published with MATLAB® R2022b

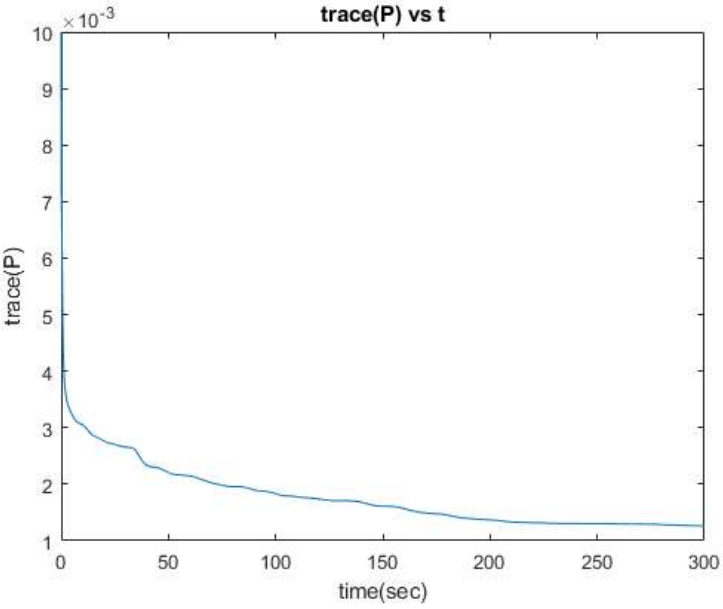
Problem 4 MATLAB Script

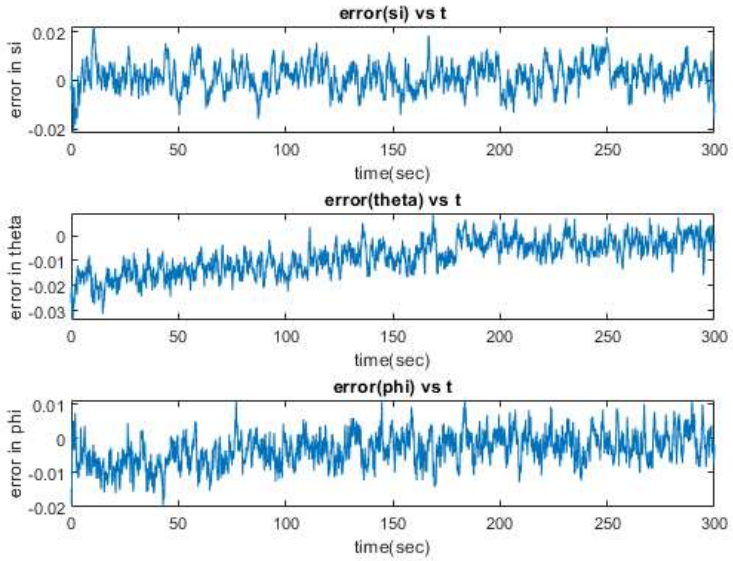
```
clear;
clc;

load ae5335_asg1_prob2_synthetic_data.mat

n_pts = numel(timestamps);
g = 9.81;

compass_cov = std_dev_compass^2;
accel_cov = (std_dev_accel^2)*eye(3);
R_cov = [compass_cov zeros(1,3); zeros(3,1) accel_cov];                    %Defining measurement error covariance
Q_cov = (std_dev_rategyro^2)*eye(3);                                       %Defining process error covariance
P_cov = eye(10)*0.001;                                                     %Initilizing estimation error covariance

%Initlizing states
si_initial = z_compass_mb(1);
theta_initial = asin(z_accelerometer_ab(1,1)/9.81);
tan_phi = z_accelerometer_ab(2,1)/z_accelerometer_ab(3,1);
phi_initial = atan(tan_phi);
x_hat = [si_initial; theta_initial; phi_initial; 0.01*randn(7, 1)];
x_int = x_hat(1:3);


%Initilizing Data matrix for plots
x_hat_plot = [];
x_int_plot = [];
P_trace_plot = [];

%Saving First data set
x_hat_plot(:,1) = x_hat;
x_int_plot(:,1) = x_int;
P_trace_plot(:,1) = trace(P_cov);

for k = 2 : n_pts

    dt = timestamps(k) - timestamps(k-1);

    si = x_hat(1);
    theta = x_hat(2);
    phi = x_hat(3);

    % Rate Gyro readings
    u_k1 = z_rategyro_rb(:,k-1);

    H0 = [0           sin(phi)              cos(phi);
          0           cos(phi)*cos(theta)   -sin(phi)*cos(theta);
          cos(theta)  sin(phi)*sin(theta)   cos(phi)*sin(theta)];
    H0 = (1/cos(theta))*H0;

    %Pridicting Estimates using nonlinear model
    u_k1_biased = u_k1 - x_hat(7:9);
    x_hat_dot = [H0*u_k1_biased;zeros(7,1)];
    x_hat_minus = x_hat + x_hat_dot * dt;

    %Linearization
    F0 = [...
        0   (u_k1_biased(2)*sin(phi)+u_k1_biased(3)*cos(phi))*tan(theta)*sec(theta)  (u_k1_biased(2)*cos(phi)-u_k1_biased(3)*sin(phi))*sec(theta);
        0   0                                                                        -(u_k1_biased(2)*sin(phi)+u_k1_biased(3)*cos(phi));
        0   (u_k1_biased(2)*sin(phi)+u_k1_biased(3)*cos(phi))*(sec(theta))^2         (u_k1_biased(2)*cos(phi)-u_k1_biased(3)*sin(phi))*tan(theta)];

    F_k1 = eye(10) + [F0 zeros(3,3) -H0 zeros(3,1); zeros(7,10)]*dt;
    G2_k1 = -[H0;zeros(7,3)]*dt;

    P_minus = F_k1*P_cov*F_k1' + G2_k1*Q_cov*G2_k1';

    %Linearization of Measurement Model
    C0 = [1   0                      0;
          0   g*cos(theta)           0;
          0   g*sin(phi)*sin(theta)  -g*cos(phi)*cos(theta);
          0   g*cos(phi)*sin(theta)  g*sin(phi)*cos(theta)];
    temp_mat = [zeros(1,6) 1; eye(3) zeros(3,4)];
    C_k1 = [C0 temp_mat];

    %Calculating Kalman Gain
    L_k = P_minus*C_k1'/(C_k1*P_minus*C_k1' + R_cov);

    %Aquiring Compass and Acclelrometer readings
```

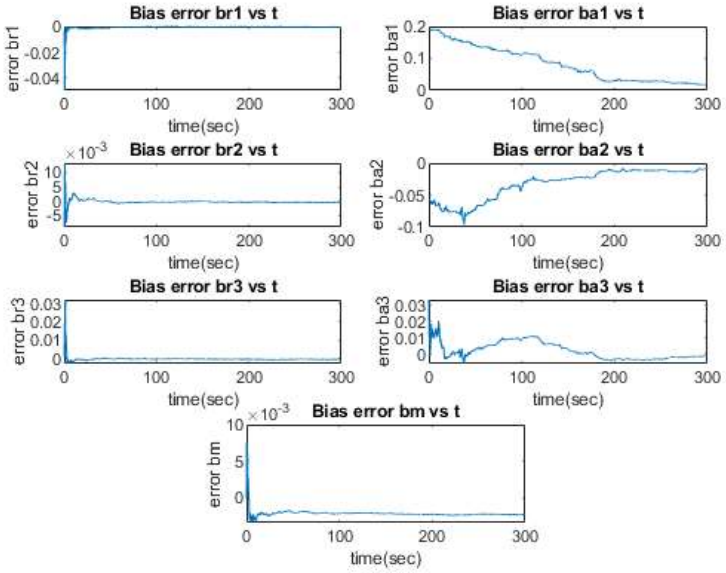
```
    z_k = [z_compass_mb(:,k);z_accelerometer_ab(:,k)];

    si = x_hat_minus(1);
    theta = x_hat_minus(2);
    phi = x_hat_minus(3);
    h_x_hat_minus = [si; g*sin(theta); -g*sin(phi)*cos(theta); -g*cos(phi)*cos(theta)] + [x_hat(10);x_hat(4:6)];

    %Updating Estimetes
    x_hat = x_hat_minus + L_k*(z_k - h_x_hat_minus);
    P_cov = (eye(10) - L_k*C_k1)*P_minus;


    %Calculating only pridictons
    H0_int = [0             sin(x_int(3))                  cos(x_int(3));
          0             cos(x_int(3))*cos(x_int(2))     -sin(x_int(3))*cos(x_int(2));
          cos(x_int(2)) sin(x_int(3))*sin(x_int(2))     cos(x_int(3))*sin(x_int(2))];
    H0_int = (1/cos(x_int(2)))*H0_int;

    x_int_dot = H0_int*u_k1;
    x_int = x_int + x_int_dot * dt;

    x_hat_plot(:,k) = x_hat;
    x_int_plot(:,k) = x_int;
    P_trace_plot(:,k) = trace(P_cov);

end

figure;
% plot 1
subplot(3,1,1)
plot(timestamps,x_hat_plot(1,:))
title('si vs t')
xlabel 'time(sec)';
ylabel 'st';
hold on;
plot(timestamps,euler_angles_true(1,:));
plot (timestamps,x_int_plot(1,:))

% plot 2
subplot(3,1,2)
plot(timestamps,x_hat_plot(2,:))
title('theta vs t')
xlabel 'time(sec)';
ylabel 'theta';
hold on;
plot(timestamps,euler_angles_true(2,:));
plot (timestamps,x_int_plot(2,:))

% plot 3
subplot(3,1,3)
plot(timestamps,x_hat_plot(3,:))
title('phi vs t')
xlabel 'time(sec)';
ylabel 'phi';
hold on;
plot(timestamps,euler_angles_true(3,:));
plot (timestamps,x_int_plot(3,:))

figure;
plot(timestamps,P_trace_plot)
title('trace(P) vs t')
xlabel 'time(sec)';
ylabel 'trace(P)';

figure;
% plot 1
subplot(3,1,1)
plot(timestamps,(euler_angles_true(1,:)-x_hat_plot(1,:)))
title('error(si) vs t')
xlabel 'time(sec)';
ylabel 'error in si';

% plot 2
subplot(3,1,2)
plot(timestamps,(euler_angles_true(2,:)-x_hat_plot(2,:)))
title('error(theta) vs t')
xlabel 'time(sec)';
ylabel 'error in theta';
```

```
% plot 3
subplot(3,1,3)
plot(timestamps,(euler_angles_true(3,:)-x_hat_plot(3,:)))
title('error(phi) vs t')
xlabel 'time(sec)';
ylabel 'error in phi';


figure;
% plot 1
subplot(4,2,1)
plot(timestamps,(bias_rategyro(1)-x_hat_plot(7,:)))
title('Bias error br1 vs t')
xlabel 'time(sec)';
ylabel 'error br1';

% plot 2
subplot(4,2,3)
plot(timestamps,(bias_rategyro(2)-x_hat_plot(8,:)))
title('Bias error br2 vs t')
xlabel 'time(sec)';
ylabel 'error br2';

% plot 3
subplot(4,2,5)
plot(timestamps,(bias_rategyro(3)-x_hat_plot(9,:)))
title('Bias error br3 vs t')
xlabel 'time(sec)';
ylabel 'error br3';

% plot 4
subplot(4,2,2)
plot(timestamps,(bias_accel(1)-x_hat_plot(4,:)))
title('Bias error ba1 vs t')
xlabel 'time(sec)';
ylabel 'error ba1';

% plot 5
subplot(4,2,4)
plot(timestamps,(bias_accel(2)-x_hat_plot(5,:)))
title('Bias error ba2 vs t')
xlabel 'time(sec)';
ylabel 'error ba2';

% plot 6
subplot(4,2,6)
plot(timestamps,(bias_accel(3)-x_hat_plot(6,:)))
title('Bias error ba3 vs t')
xlabel 'time(sec)';
ylabel 'error ba3';

% plot 7
subplot(4,2,7.5)
plot(timestamps,(bias_compass-x_hat_plot(10,:)))
title('Bias error bm vs t')
xlabel 'time(sec)';
ylabel 'error bm';
```


si vs t
st
time(sec)
theta vs t
theta
time(sec)
phi vs t
phi
time(sec)


trace(P) vs t
×10^-3
trace(P)
time(sec)





Published with MATLAB® R2022b

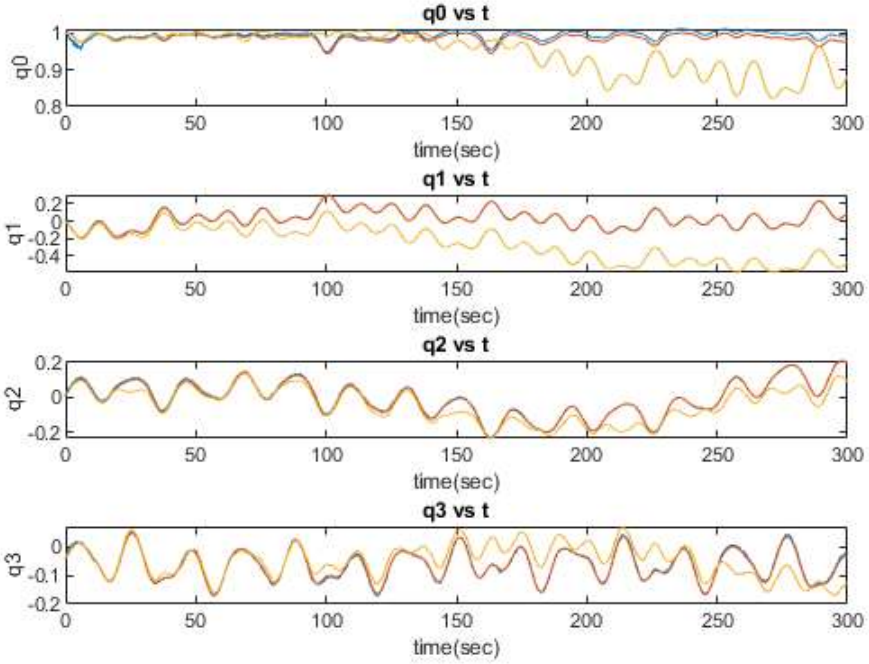
Problem 5 MATLAB Script

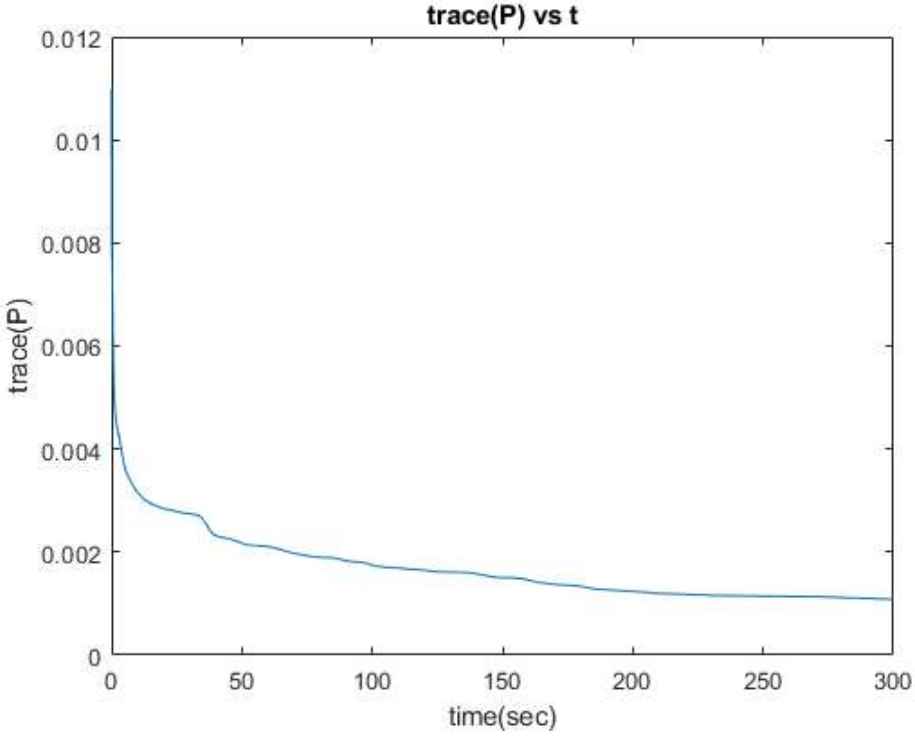
```
clear;
clc;

load ae5335_asg1_prob2_synthetic_data.mat

n_pts = numel(timestamps);
g = 9.81;

compass_cov = std_dev_compass^2;
accel_cov = (std_dev_accel^2)*eye(3);
R_cov = [compass_cov zeros(1,3); zeros(3,1) accel_cov];                    %Defining measurement error covariance
Q_cov = (std_dev_rategyro^2)*eye(3);                                       %Defining process error covariance
P_cov = eye(11)*0.001;                                                     %Initilizing estimation error covariance

%Initlizing states
std_dev_noise = 0.01;
initial_measurements = quaternions_true(:,1) + std_dev_noise^2*randn(4,1);
x_hat = [initial_measurements; 0.001*randn(7, 1)];
x_int = x_hat(1:4);


%Initilizing Data matrix for plots
x_hat_plot = [];
x_int_plot = [];
P_trace_plot = [];
mod_q_plot = [];

%Saving First data set
x_hat_plot(:,1) = x_hat;
x_int_plot(:,1) = x_int;
P_trace_plot(:,1) = trace(P_cov);
mod_q_plot(:,1) = sqrt(x_hat(1)^2+x_hat(2)^2+x_hat(3)^2+x_hat(4)^2);

for k = 2 : n_pts

    dt = timestamps(k) - timestamps(k-1);

    e0 = x_hat(1);
    e1 = x_hat(2);
    e2 = x_hat(3);
    e3 = x_hat(4);

    % Rate Gyro readings
    u_k1 = z_rategyro_rb(:,k-1);

    H0 = 0.5 *  [-e1 -e2 -e3;
                  e0 -e3  e2;
                  e3  e0 -e1;
                 -e2  e1  e0];

    %Pridicting Estimates
    u_k1_biased = u_k1 - x_hat(8:10);
    x_hat_dot = [H0*u_k1_biased; zeros(7,1)];
    x_hat_minus = x_hat + x_hat_dot * dt;

    %Linearization
    p = u_k1_biased(1);
    q = u_k1_biased(2);
    r = u_k1_biased(3);
    F0 = 0.5 * [0 -p -q -r;
                p  0  r -q;
                q  -r  0  p;
```

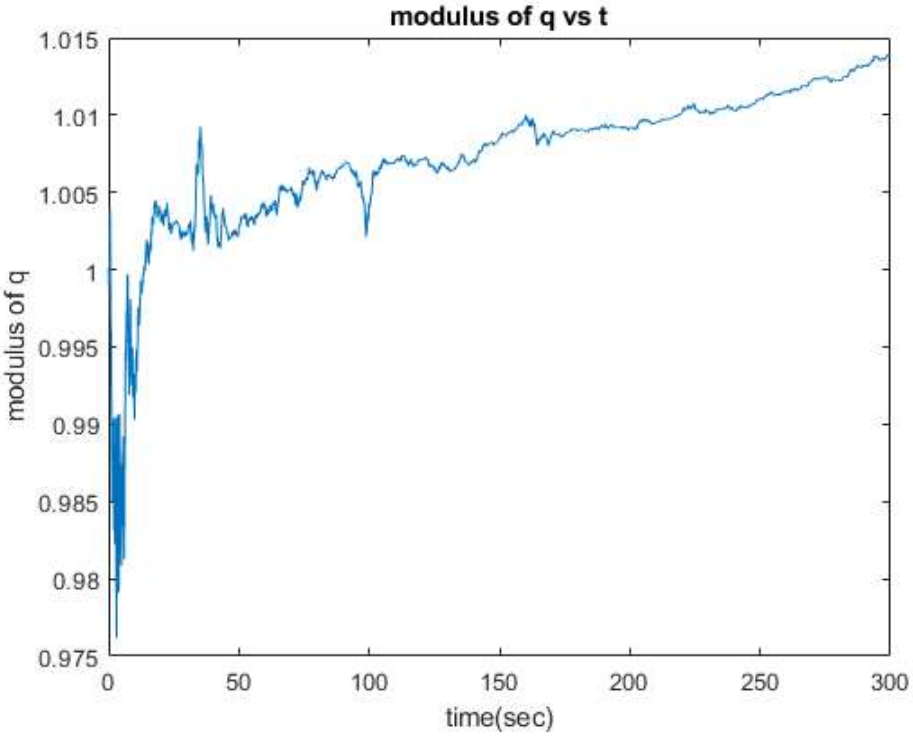
```
                r  q -p  0];
    F_k1 = eye(11) + [F0 zeros(4,3) -H0 zeros(4,1); zeros(7,11)]*dt;
    G2_k1 = -[H0;zeros(7,3)]*dt;

    P_minus = F_k1*P_cov*F_k1' + G2_k1*Q_cov*G2_k1';

    %Linearization of Measurement Model
    Y = 2*(e0*e3 + e1*e2);
    X = e0^2 + e1^2 - e2^2 - e3^2;
    si_dot_e0 = (2*e3*X - 2*e0*Y) / (X^2+Y^2);
    si_dot_e1 = (2*e2*X - 2*e1*Y) / (X^2+Y^2);
    si_dot_e2 = (2*e1*X + 2*e2*Y) / (X^2+Y^2);
    si_dot_e3 = (2*e0*X + 2*e3*Y) / (X^2+Y^2);

    C0 = [si_dot_e0  si_dot_e1  si_dot_e2  si_dot_e3;
            2*g*e2    -2*g*e3     2*g*e0    -2*g*e1;
           -2*g*e1    -2*g*e0    -2*g*e3    -2*g*e2;
           -0*g*e0     4*g*e1     4*g*e2    -0*g*e3];
    temp_mat = [zeros(1,6) 1; eye(3) zeros(3,4)];
    C_k1 = [C0 temp_mat];

    %Calculating Kalman Gain
    L_k = P_minus*C_k1'/(C_k1*P_minus*C_k1' + R_cov);

    %Aquiring Compass and Acclelrometer readings
    z_k = [z_compass_mb(:,k);z_accelerometer_ab(:,k)];

    e0 = x_hat_minus(1);
    e1 = x_hat_minus(2);
    e2 = x_hat_minus(3);
    e3 = x_hat_minus(4);
    Y = 2*(e0*e3 + e1*e2);
    X = e0^2 + e1^2 - e2^2 - e3^2;
    si = atan2(Y,X);
    h_x_hat_minus = [si; -2*g*(-e0*e2 + e1*e3); -2*g*(e0*e1 + e2*e3); -g*(1 -2*(e1^2 + e2^2))] + [x_hat(11);x_hat(5:7)];

    %Updating Estimates
    x_hat = x_hat_minus + L_k*(z_k - h_x_hat_minus);
    P_cov = (eye(11) - L_k*C_k1)*P_minus;


    %Calculating only pridictons
    p = u_k1(1);
    q = u_k1(2);
    r = u_k1(3);
    F0 = 0.5 * [0 -p -q -r;
                p  0  r -q;
                q  r  0  p;
                r  q -p  0];
    x_int_dot = F0*x_int;
    x_int = x_int + x_int_dot * dt;

    x_hat_plot(:,k) = x_hat;
    x_int_plot(:,k) = x_int;
    P_trace_plot(:,k) = trace(P_cov);
    mod_q_plot(:,k) = sqrt(x_hat(1)^2+x_hat(2)^2+x_hat(3)^2+x_hat(4)^2);

end

figure;
% plot 1
subplot(4,1,1)
plot(timestamps,x_hat_plot(1,:))
title('q0 vs t')
```

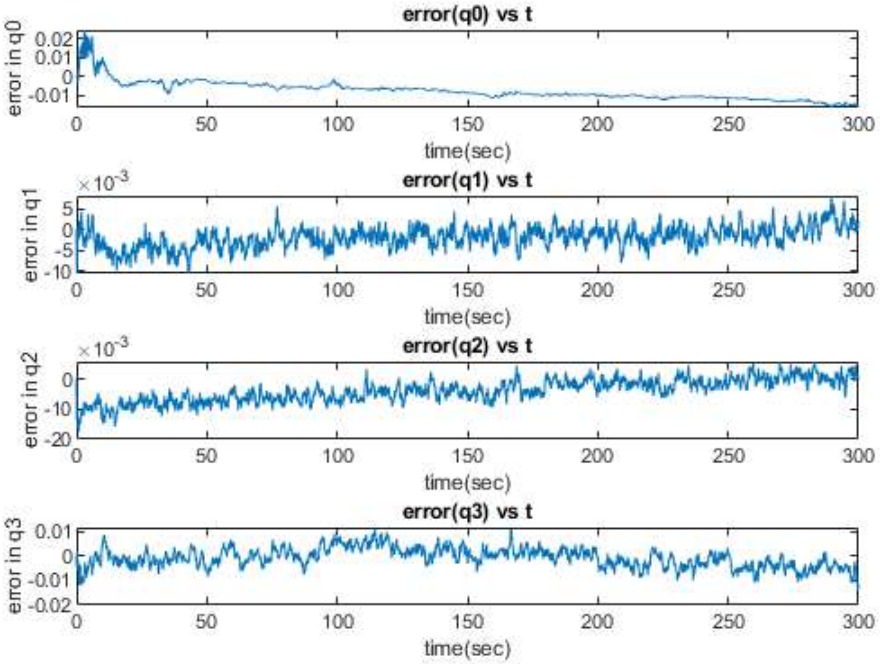
```
xlabel 'time(sec)';
ylabel 'q0';
hold on;
plot(timestamps,quaternions_true(1,:));
plot (timestamps,x_int_plot(1,:))

% plot 2
subplot(4,1,2)
plot(timestamps,x_hat_plot(2,:))
title('q1 vs t')
xlabel 'time(sec)';
ylabel 'q1';
hold on;
plot(timestamps,quaternions_true(2,:));
plot (timestamps,x_int_plot(2,:))

% plot 3
subplot(4,1,3)
plot(timestamps,x_hat_plot(3,:))
title('q2 vs t')
xlabel 'time(sec)';
ylabel 'q2';
hold on;
plot(timestamps,quaternions_true(3,:));
plot (timestamps,x_int_plot(3,:))

%plot4
subplot(4,1,4)
plot(timestamps,x_hat_plot(4,:))
title('q3 vs t')
xlabel 'time(sec)';
ylabel 'q3';
hold on;
plot(timestamps,quaternions_true(4,:));
plot (timestamps,x_int_plot(4,:))

figure;
plot(timestamps,P_trace_plot)
title('trace(P) vs t')
xlabel 'time(sec)';
ylabel 'trace(P)';

figure;
plot(timestamps,mod_q_plot)
title('modulus of q vs t')
xlabel 'time(sec)';
ylabel 'modulus of q';

figure;
% plot 1
subplot(4,1,1)
plot(timestamps,(quaternions_true(1,:)-x_hat_plot(1,:)))
title('error(q0) vs t')
xlabel 'time(sec)';
ylabel 'error in q0';

% plot 2
subplot(4,1,2)
plot(timestamps,(quaternions_true(2,:)-x_hat_plot(2,:)))
title('error(q1) vs t')
xlabel 'time(sec)';
ylabel 'error in q1';

% plot 3
```

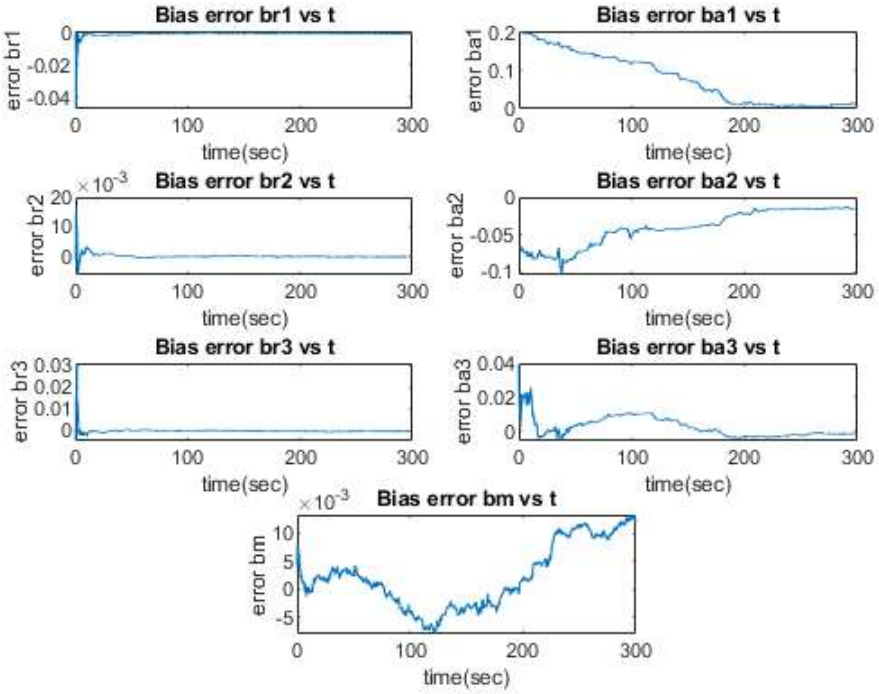
```
subplot(4,1,3)
plot(timestamps,(quaternions_true(3,:)-x_hat_plot(3,:)))
title('error(q2) vs t')
xlabel 'time(sec)';
ylabel 'error in q2';

% plot 4
subplot(4,1,4)
plot(timestamps,(quaternions_true(4,:)-x_hat_plot(4,:)))
title('error(q3) vs t')
xlabel 'time(sec)';
ylabel 'error in q3';


figure;
% plot 1
subplot(4,2,1)
plot(timestamps,(bias_rategyro(1)-x_hat_plot(8,:)))
title('Bias error br1 vs t')
xlabel 'time(sec)';
ylabel 'error br1';

% plot 2
subplot(4,2,3)
plot(timestamps,(bias_rategyro(2)-x_hat_plot(9,:)))
title('Bias error br2 vs t')
xlabel 'time(sec)';
ylabel 'error br2';

% plot 3
subplot(4,2,5)
plot(timestamps,(bias_rategyro(3)-x_hat_plot(10,:)))
title('Bias error br3 vs t')
xlabel 'time(sec)';
ylabel 'error br3';

% plot 4
subplot(4,2,2)
plot(timestamps,(bias_accel(1)-x_hat_plot(5,:)))
title('Bias error ba1 vs t')
xlabel 'time(sec)';
ylabel 'error ba1';

% plot 5
subplot(4,2,4)
plot(timestamps,(bias_accel(2)-x_hat_plot(6,:)))
title('Bias error ba2 vs t')
xlabel 'time(sec)';
ylabel 'error ba2';

% plot 6
subplot(4,2,6)
plot(timestamps,(bias_accel(3)-x_hat_plot(7,:)))
title('Bias error ba3 vs t')
xlabel 'time(sec)';
ylabel 'error ba3';

% plot 7
subplot(4,2,7.5)
plot(timestamps,(bias_compass-x_hat_plot(11,:)))
title('Bias error bm vs t')
xlabel 'time(sec)';
ylabel 'error bm';
```


q0 vs t
q0
time(sec)
q1 vs t
q1
time(sec)
q2 vs t
q2
time(sec)
q3 vs t
q3
time(sec)


trace(P) vs t
trace(P)
time(sec)


modulus of q vs t
modulus of q
time(sec)
1.015
1.01
1.005
1
0.995
0.99
0.985
0.98
0.975
0
50
100
150
200
250
300


error(q0) vs t
error in q0
time(sec)
error(q1) vs t
error in q1
×10^-3
error(q2) vs t
error in q2
×10^-3
error(q3) vs t
error in q3



Published with MATLAB® R2022b