EE5114 Autonomous Robot Navigation

EE5114 Continuous Assessment 1

NAME: MADKAIKAR ATHARVA ATUL

MATRICULATION NO: A0268376M

1. By understanding the original codes and observing the default plots of this set of data, can you deduce how many times the UAV had taken off and landed?

By observing the GPS and the Motor Signal plots, we can deduce that the UAV had taken off and landed twice.

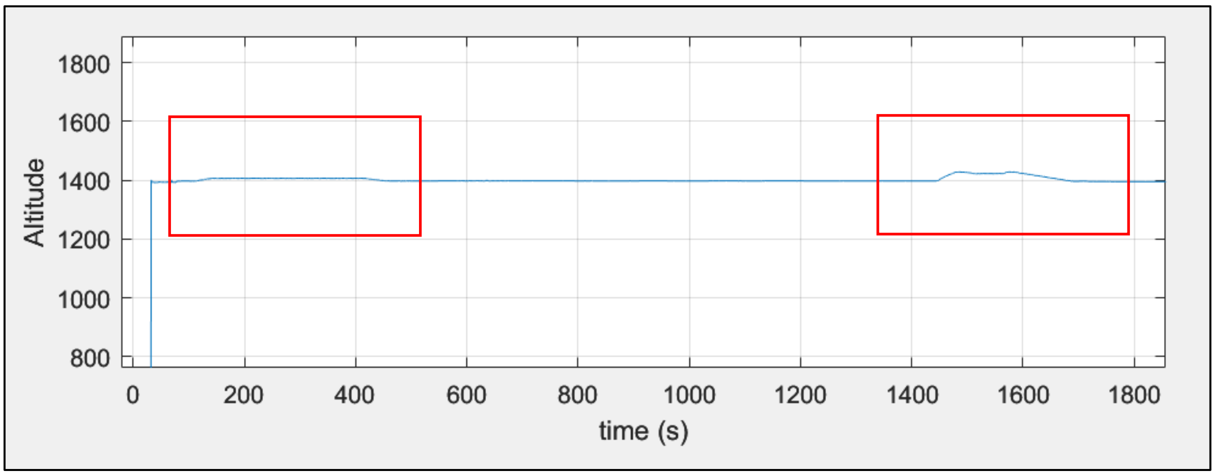


Figure 1: GPS Plot – Altitude

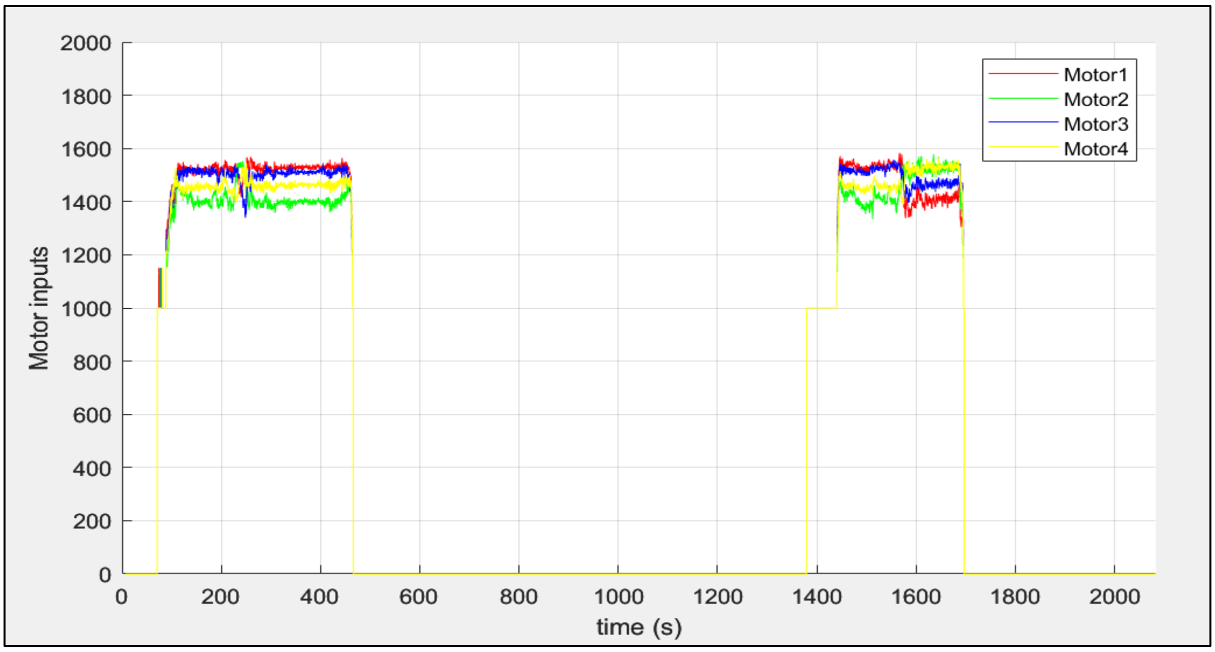


Figure 2: Motor Signal Plots

2. When did the UAV take off and land for the 2nd time? Please provide relevant figures to support your explanations.

By zooming in on the GPS plot for altitude and observing the motor signal plots, we can see that the UAV took off for the second time at around 1444.66 seconds and landed at around 1688.78 seconds. It is also evident from the motor signal plot that all the motors are powered on at around the same.

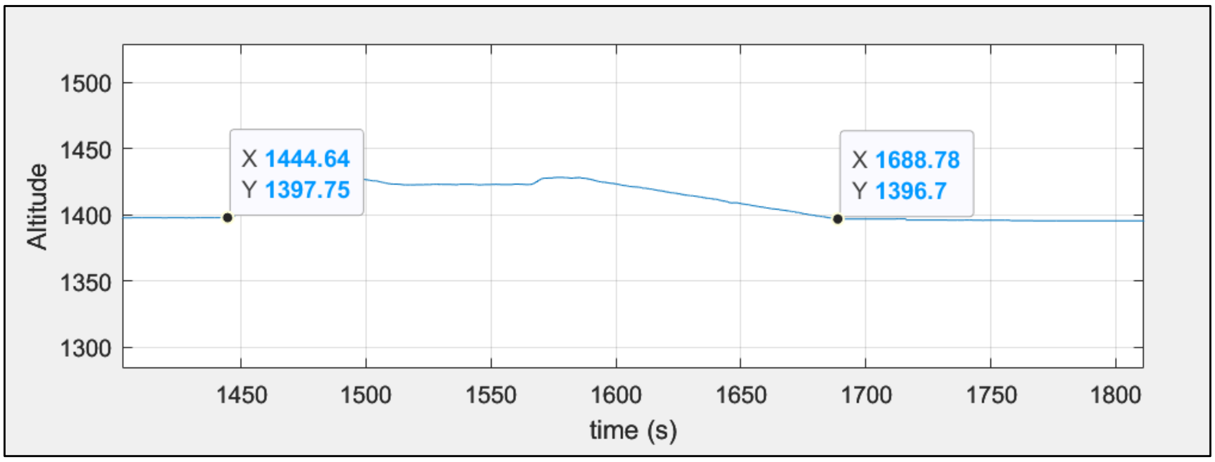


Figure 3: GPS Plot – Altitude

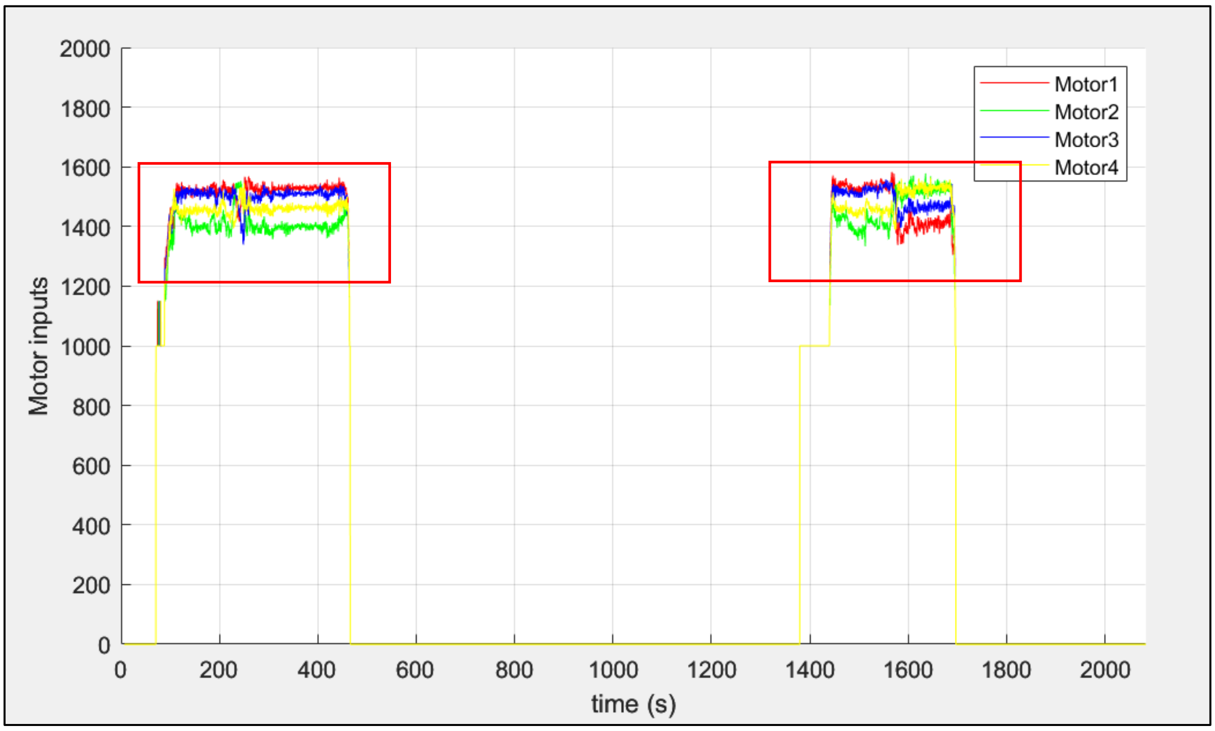


Figure 4: Motor Signal Plots

3. List the formulas you have used to convert GPS coordinates to NED positions and attach your corresponding codes for this step.

GPS co-ordinates are expressed in Geodetic co-ordinate system. We first need to convert Geodetic co-ordinates to ECEF co-ordinates followed by conversion to NED co-ordinates. The conversion formulae have been listed below:

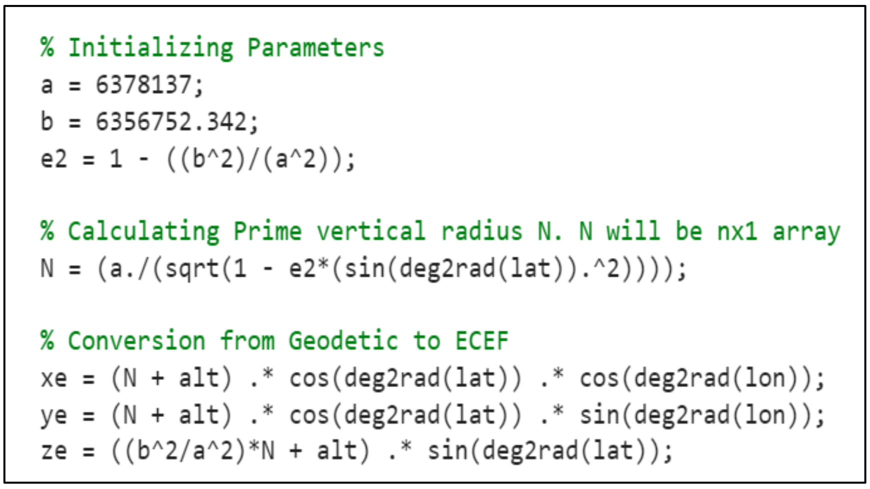
a. Conversion from Geodetic to ECEF co-ordinates:

$$\begin{cases} X_e = (N(\varphi) + h)\cos\varphi\cos\lambda \\ Y_e = (N(\varphi) + h)\cos\varphi\sin\lambda \\ Z_e = \left(\frac{b^2}{a^2}\mathrm{N}(\varphi) + \mathrm{h}\right)\sin\varphi \end{cases}$$

where $N(\varphi) = \dfrac{a^2}{\sqrt{a^2\cos^2\varphi + b^2\sin^2\varphi}} = \dfrac{a^2}{\sqrt{1 - e^2\sin^2\varphi}}$

a: Equatorial radius (semi-major axis)
b: Polar radius (semi-minor axis)
$e^2 = 1 - \frac{b^2}{a^2}$: Square of the first numerical eccentricity of ellipsoid
$N(\varphi)$: Prime vertical radius of curvature

<u>MATLAB CODE:</u>



```
% Initializing Parameters
a = 6378137;
b = 6356752.342;
e2 = 1 - ((b^2)/(a^2));

% Calculating Prime vertical radius N. N will be nx1 array
N = (a./(sqrt(1 - e2*(sin(deg2rad(lat)).^2))));

% Conversion from Geodetic to ECEF
xe = (N + alt) .* cos(deg2rad(lat)) .* cos(deg2rad(lon));
ye = (N + alt) .* cos(deg2rad(lat)) .* sin(deg2rad(lon));
ze = ((b^2/a^2)*N + alt) .* sin(deg2rad(lat));
```

Figure 5: MATLAB code, Geodetic to ECEF

b. Conversion from ECEF to Local NED Co-ordinates:

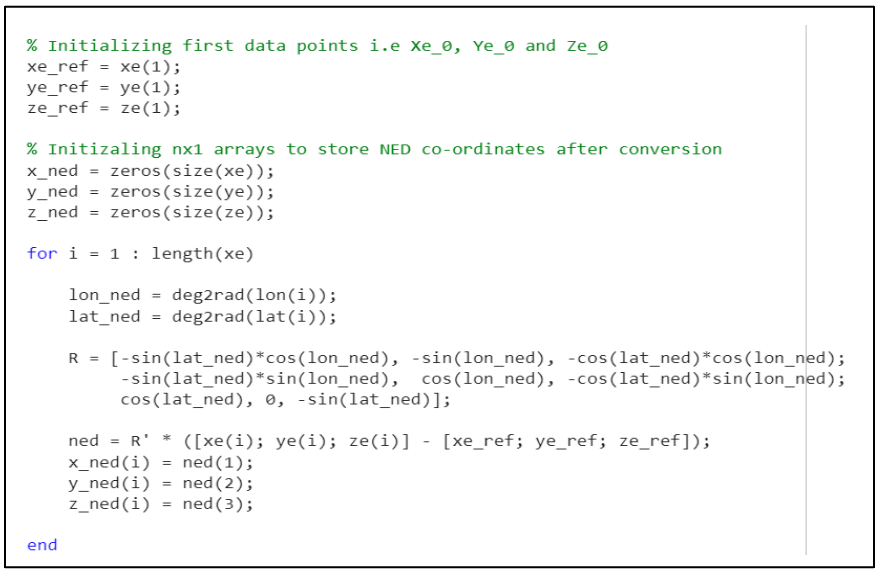
$$\begin{bmatrix} X_n \\ Y_n \\ Z_n \end{bmatrix} = R^T\left(\begin{bmatrix} X_e \\ Y_e \\ Z_e \end{bmatrix} - \begin{bmatrix} X_{e0} \\ Y_{e0} \\ Z_{e0} \end{bmatrix}\right)$$

(Zero) reference point
in ECEF coordinates

$$R = \begin{bmatrix} -\sin\varphi\cos\lambda & -\sin\lambda & -\cos\varphi\cos\lambda \\ -\sin\varphi\sin\lambda & \cos\lambda & -\cos\varphi\sin\lambda \\ \cos\varphi & 0 & -\sin\varphi \end{bmatrix}$$

Where, R is the Rotation matrix that converts ECEF Co-ordinates to NED Co-ordinates.

MATLAB CODE:



```
% Initializing first data points i.e Xe_0, Ye_0 and Ze_0
xe_ref = xe(1);
ye_ref = ye(1);
ze_ref = ze(1);

% Initizaling nx1 arrays to store NED co-ordinates after conversion
x_ned = zeros(size(xe));
y_ned = zeros(size(ye));
z_ned = zeros(size(ze));

for i = 1 : length(xe)

    lon_ned = deg2rad(lon(i));
    lat_ned = deg2rad(lat(i));

    R = [-sin(lat_ned)*cos(lon_ned), -sin(lon_ned), -cos(lat_ned)*cos(lon_ned);
         -sin(lat_ned)*sin(lon_ned),  cos(lon_ned), -cos(lat_ned)*sin(lon_ned);
         cos(lat_ned), 0, -sin(lat_ned)];

    ned = R' * ([xe(i); ye(i); ze(i)] - [xe_ref; ye_ref; ze_ref]);
    x_ned(i) = ned(1);
    y_ned(i) = ned(2);
    z_ned(i) = ned(3);

end
```

Figure 6: MATLAB code, ECEF to NED

After conversion to ECEF, the first data point (X, Y, Z) in ECEF co-ordinate system has been taken as the Zero Reference Point for further conversion to NED Co-ordinates.

4. List the formulas you have used for the implementation of Kalman filter and attach your corresponding codes for this step

The formulae used are as follows:

Process Model and Prediction step

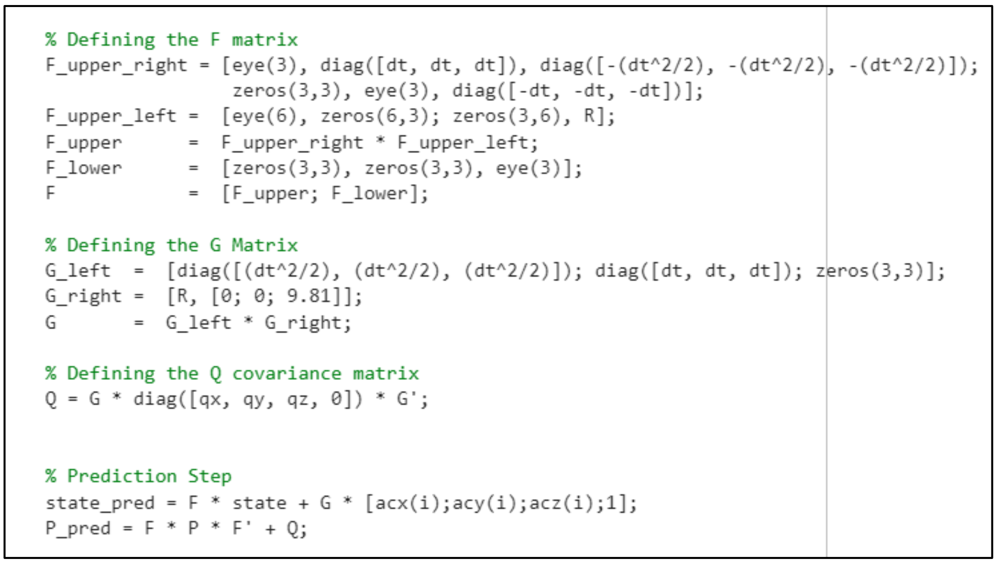
$$\begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ b_x \\ b_y \\ b_z \end{bmatrix}_k = \overbrace{\begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 & -dt^2/2 & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 & 0 & -dt^2/2 & 0 \\ 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 & -dt^2/2 \\ 0 & 0 & 0 & 1 & 0 & 0 & -dt & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -dt & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -dt \end{bmatrix} \begin{bmatrix} I_6 & 0 \\ 0 & \boldsymbol{R}_{g/b} \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{bmatrix}}^{F} \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ b_x \\ b_y \\ b_z \end{bmatrix}_{k-1} + \overbrace{\begin{bmatrix} dt^2/2 & 0 & 0 \\ 0 & dt^2/2 & 0 \\ 0 & 0 & dt^2/2 \\ dt & 0 & 0 \\ 0 & dt & 0 \\ 0 & 0 & dt \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_{g/b} & \begin{matrix} 0 \\ 0 \\ g \end{matrix} \end{bmatrix}}^{G} \begin{bmatrix} a_x \\ a_y \\ a_z \\ 1 \end{bmatrix}_k$$

$$\boldsymbol{R_{g/b}} = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix}$$

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1})$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^{\mathrm{T}} + \mathbf{Q}_{\mathrm{cov}}$$

MATLAB Code:



```
% Defining the F matrix
F_upper_right = [eye(3), diag([dt, dt, dt]), diag([-(dt^2/2), -(dt^2/2), -(dt^2/2)]);
                 zeros(3,3), eye(3), diag([-dt, -dt, -dt])];
F_upper_left =  [eye(6), zeros(6,3); zeros(3,6), R];
F_upper       =  F_upper_right * F_upper_left;
F_lower       =  [zeros(3,3), zeros(3,3), eye(3)];
F             =  [F_upper; F_lower];

% Defining the G Matrix
G_left  =  [diag([(dt^2/2), (dt^2/2), (dt^2/2)]); diag([dt, dt, dt]); zeros(3,3)];
G_right =  [R, [0; 0; 9.81]];
G       =  G_left * G_right;

% Defining the Q covariance matrix
Q = G * diag([qx, qy, qz, 0]) * G';


% Prediction Step
state_pred = F * state + G * [acx(i);acy(i);acz(i);1];
P_pred = F * P * F' + Q;
```

Figure 7: MATLAB code, Process Model and Prediction

Measurement Model Correction Step:

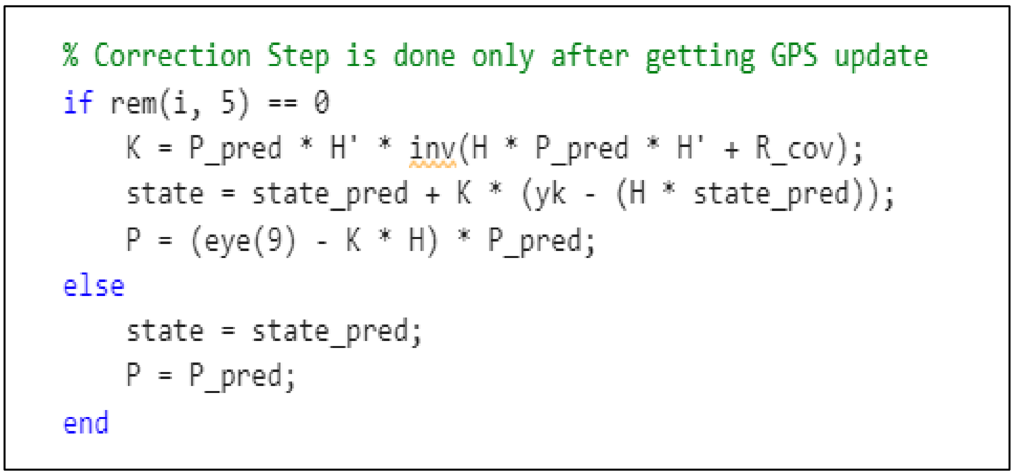
$$\boxed{[\mathbf{p}] = [\mathbf{I} \quad 0 \quad 0]\begin{bmatrix}\mathbf{p}\\ \mathbf{v}\\ \mathbf{b}\end{bmatrix}}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_{\text{cov}})^{-1},$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1}),$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1},$$

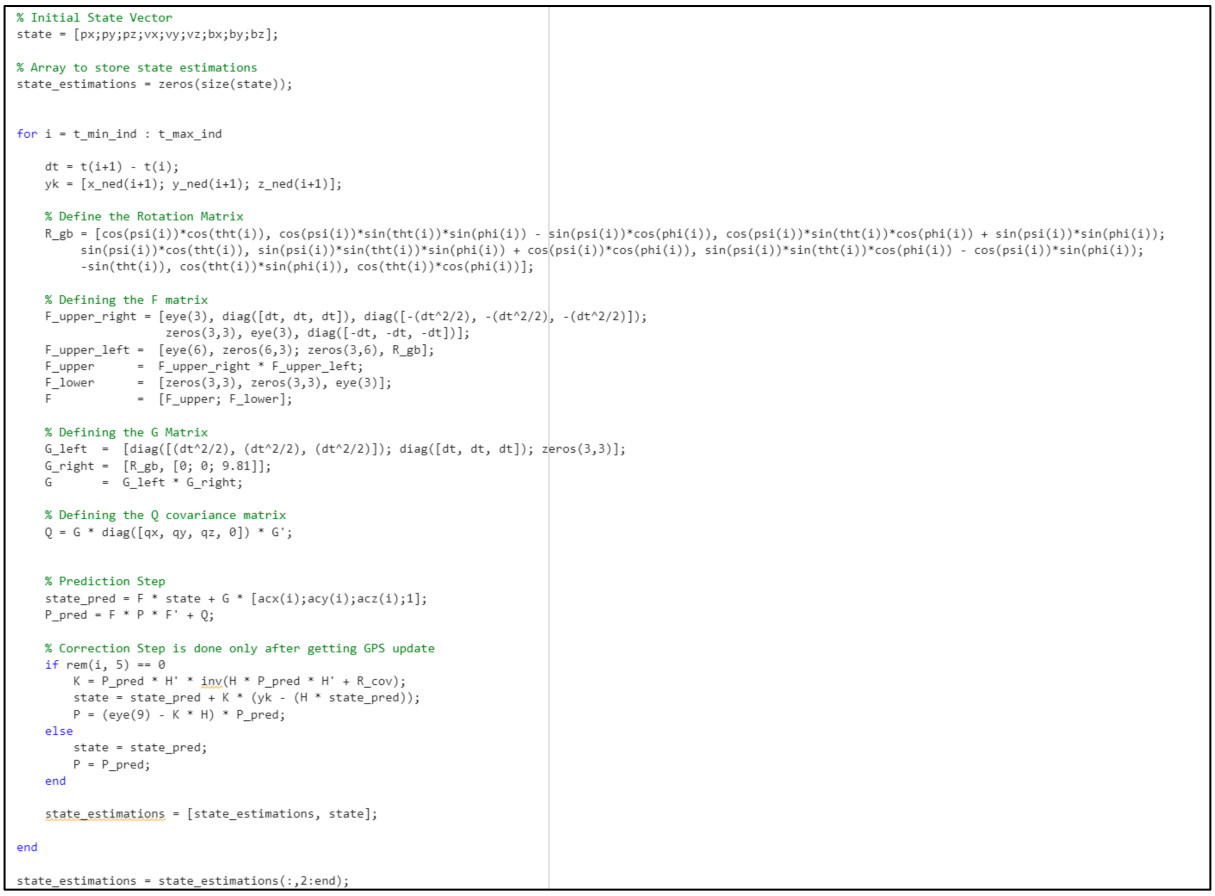
MATLAB Code:



```
% Correction Step is done only after getting GPS update
if rem(i, 5) == 0
    K = P_pred * H' * inv(H * P_pred * H' + R_cov);
    state = state_pred + K * (yk - (H * state_pred));
    P = (eye(9) - K * H) * P_pred;
else
    state = state_pred;
    P = P_pred;
end
```

Figure 8: MATLAB code, Measurement Model and Correction

MATLAB Code for EKF:



```matlab
% Initial State Vector
state = [px;py;pz;vx;vy;vz;bx;by;bz];

% Array to store state estimations
state_estimations = zeros(size(state));


for i = t_min_ind : t_max_ind

    dt = t(i+1) - t(i);
    yk = [x_ned(i+1); y_ned(i+1); z_ned(i+1)];

    % Define the Rotation Matrix
    R_gb = [cos(psi(i))*cos(tht(i)), cos(psi(i))*sin(tht(i))*sin(phi(i)) - sin(psi(i))*cos(phi(i)), cos(psi(i))*sin(tht(i))*cos(phi(i)) + sin(psi(i))*sin(phi(i));
          sin(psi(i))*cos(tht(i)), sin(psi(i))*sin(tht(i))*sin(phi(i)) + cos(psi(i))*cos(phi(i)), sin(psi(i))*sin(tht(i))*cos(phi(i)) - cos(psi(i))*sin(phi(i));
          -sin(tht(i)), cos(tht(i))*sin(phi(i)), cos(tht(i))*cos(phi(i))];

    % Defining the F matrix
    F_upper_right = [eye(3), diag([dt, dt, dt]), diag([-(dt^2/2), -(dt^2/2), -(dt^2/2)]);
                      zeros(3,3), eye(3), diag([-dt, -dt, -dt])];
    F_upper_left =  [eye(6), zeros(6,3); zeros(3,6), R_gb];
    F_upper      =  F_upper_right * F_upper_left;
    F_lower      =  [zeros(3,3), zeros(3,3), eye(3)];
    F            =  [F_upper; F_lower];

    % Defining the G Matrix
    G_left  =  [diag([(dt^2/2), (dt^2/2), (dt^2/2)]); diag([dt, dt, dt]); zeros(3,3)];
    G_right =  [R_gb, [0; 0; 9.81]];
    G       =  G_left * G_right;

    % Defining the Q covariance matrix
    Q = G * diag([qx, qy, qz, 0]) * G';


    % Prediction Step
    state_pred = F * state + G * [acx(i);acy(i);acz(i);1];
    P_pred = F * P * F' + Q;

    % Correction Step is done only after getting GPS update
    if rem(i, 5) == 0
        K = P_pred * H' * inv(H * P_pred * H' + R_cov);
        state = state_pred + K * (yk - (H * state_pred));
        P = (eye(9) - K * H) * P_pred;
    else
        state = state_pred;
        P = P_pred;
    end

    state_estimations = [state_estimations, state];

end

state_estimations = state_estimations(:,2:end);
```

Figure 9: MATLAB code, EKF

5. Please explain the physical meanings of your state variables used in your EKF and which coordinate system they are defined in.

The state variables used in EKF represent the estimated dynamic characteristics of a system being tracked or estimated. The physical meanings of the state variables in the context of these assignments are as follows:

a) Position variables (px, py, pz):
These represent the estimated position of the object in space. In the assignment, these variables are defined in the NED co-ordinate system.

b) Velocity variables (vx, vy, vz):
These represent the estimated velocities of the object being tracked. In the assignment, these variables are defined in the NED co-ordinate system.

c) Acceleration biases (bx, by, bz):
Acceleration biases represent the errors or offsets in the sensor measurement (Accelerometer readings). These indicate the deviation of sensor readings from true acceleration values when the sensor is at rest. In the assignment, these variables are defined in the body frame system.

6. Note that GPS update rate is slower than IMU update rate for this set of data. Please explain how you have implemented your code to address this practical issue.

In the dataset, the accelerometer updates occur roughly every 0.2 seconds, while the GPS updates are received roughly once every second. This means that the accelerometer provides data five times more frequently than the GPS. To account for this difference in update rates, the correction step should be executed whenever a GPS update is available. Since the GPS updates occur every 5-time steps (1 second / 0.2 seconds = 5), the code should perform the correction step at intervals that are multiples of 5-time steps.

Also, it's important to note that even if the correction step is not executed due to the absence of GPS updates at a particular time step, the state and covariance will still be updated based on the accelerometer data during the prediction step. This ensures that the state estimate evolves continuously over time, incorporating the information from the accelerometer measurements. The correction step, when executed, further refines the state estimate based on the more accurate GPS measurements and updates the covariance accordingly.

7. Please explain what values you have chosen to initialize the state variables and the state covariance matrix P. Why these values?

The EKF was implemented between t_min (10 minutes before the second flight) and t_max (5 minutes after the UAV landed). To initialize the state variables effectively, NED coordinates and velocities at t = t_min were utilized, providing a stable reference point for the estimation process, and ensuring a better initialization for the EKF.

The accelerometer biases (bx, by, and bz) were initialized to zero because the UAV was observed to be at rest at the time of initialization, and the z-component of acceleration (acz) provided a reading close to the expected gravitational acceleration (-9.81 m/s^2). This indicated that the accelerometer sensor already accounted for gravitational acceleration, making bias correction unnecessary in the initial state estimation.

The initialization values were:

- **px** = 3.077×10^5, **py** = -3.7638×10^4, **pz** = -8.6940×10^3
- **vx** = 0, **vy** = 0, **vz** = 0
- **bx** = 0, **by** = 0, **bz** = 0

The state covariance matrix ‘P’ was initialized to a 9 x 9 diagonal matrix with smaller values for position components and larger values for velocity and acceleration bias components. Initializing P in such a way reflects confidence in the initial position but uncertainty about the velocity estimates and bias terms, which is case with the data provided to us. **So, the matrix was initialized to:**

P = diag ([0.1, 0.1, 0.1, 10, 10, 10, 5, 5, 5]);

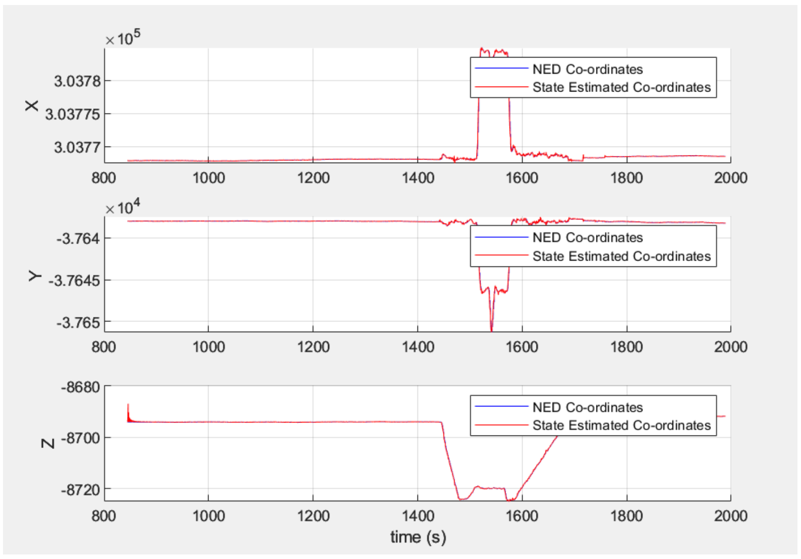


Figure 10: NED and Estimated Coordinates when P = diag ([0.1, 0.1, 0.1, 10, 10, 10, 5, 5, 5])

Based on the results seen in the above figure, it is evident that initializing the state covariance matrix P with the above specified values results in accurate and reliable state estimates.

Moreover, initializing values based on the confidence we have Position, Velocity and Bias estimates helps the EKF's performance and convergence as against initializing P to just a diagonal matrix multiplied by a scalar value as evidenced from the results below.

*P = eye (9) * 0.01*

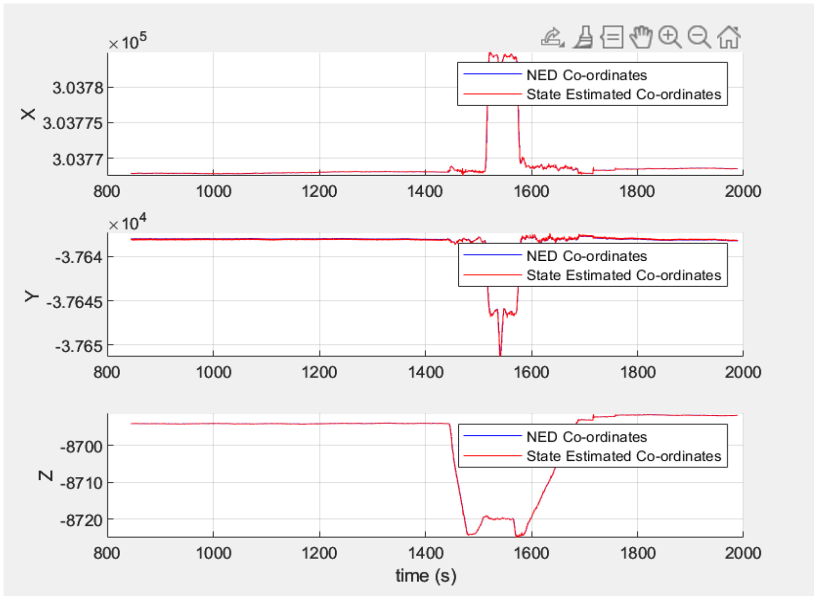


*Figure 11: Estimated Co-ordinates for P = eye (9) * 0.01*

The plot of estimations over time clearly illustrates that initializing the state covariance matrix P as a diagonal matrix scaled by a scalar value leads to noticeable noise in the estimation data.

8. Please explain your choice of Q and R matrices for your EKF implementation.

The Process Covariance Matrix Q was calculated using the formula provided in the lecture notes:

$$Q = G \begin{bmatrix} q_x & 0 & 0 & 0 \\ 0 & q_y & 0 & 0 \\ 0 & 0 & q_z & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} G'$$

Where, qx, qy and qz are tuneable parameters representing the process noise in the x, y, and z states respectively. Experimenting with these parameters, it was seen that EKF performs well when these parameters are sufficiently large. The following values were assigned in the code:

- qx = 50
- qy = 75
- qz = 100

Since, we have a good estimation of the initial position of the UAV, the measurement covariance matrix 'R' was initialized with smaller values to reflect the confidence in the GPS measurement.

The measurement covariance matrix R was initialized to a 3x3 identity matrix multiplied by a scalar value.

Thus**, R = 3x3 identity Matrix * 1** was used in the final code.

9. Have the accelerometer bias values converged in the end? What values did they converge on?

Yes, the accelerometer bias values are converging in the end as can be seen from the plots.

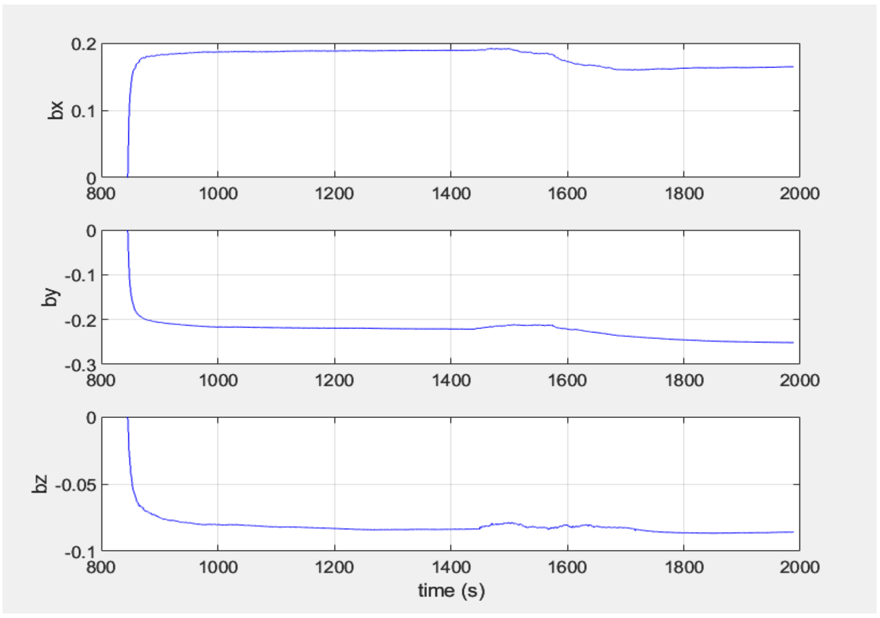


Figure 12:Estimated Acceleration Biases

The biases converged on the following values:

- 0.1648, -0.2516 and -0.0858 along the X, Y and Z axes respectively.

10. If considering accelerometer bias is not a constant, but gradually drifts over time, how will you modify the implementation of EKF to address this issue?

If the biases are not constant and drift over time, modifications can be made to the state transition matrix F to include the dynamics of the bias terms. Moreover, sensor fusion using more sensors can help with the drifting issue as well.