EE5114 - Autonomous Robot Navigation

National University of Singapore

Department of Mechanical Engineering

EE5114-CA1

Author:
LIU XIAO(ID: A0304126U)

Date: September 23, 2024

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?

In Figure 1, the GPS position curve shows changes in the drone's position during two distinct time intervals, indicating that the drone took off and landed twice.

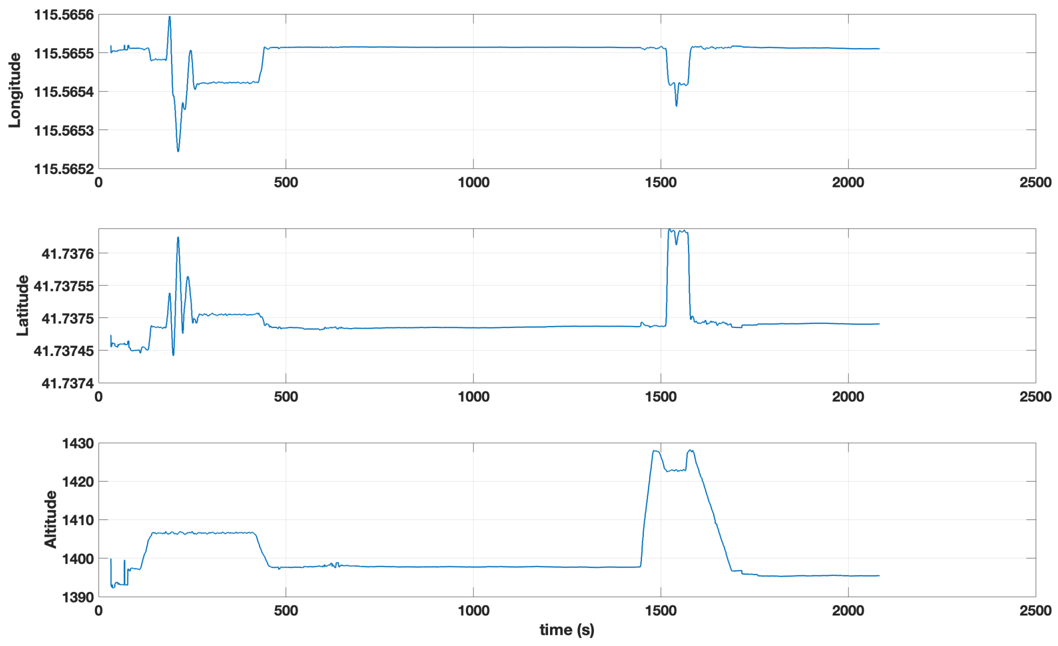

Figure 1: GPS Position

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

For the second takeoff, the UAV lifted off around 1440 seconds and landed around 1696 seconds, as shown in Figure 1, where the GPS position changes during this time interval. This is further supported by the acceleration curve in Figure 2. Therefore, t_{min} and t_{max} are 840 seconds and 1996 seconds, respectively.

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

Firstly, I transformed GPS coordinates to Earth-Centered Earth-Fixed(ECEF) frame. The formula is:

$$\begin{cases} X_{ecef} = (N(\varphi)+h)\cos\varphi\cos\lambda \\ Y_{ecef} = (N(\varphi)+h)\cos\varphi\sin\lambda \\ Z_{ecef} = \left(\frac{b^2}{a^2}\mathrm{N}(\varphi)+\mathrm{h}\right)\sin\varphi \end{cases} \tag{1}$$

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

$$e^2 = 1 - \frac{b^2}{a^2} \tag{3}$$

where a is equatorial radius of earth(6378137 m), b is polar radius of earth(6356752.31424518m) according to WGS84 standard, and φ, λ, h are the longitude, latitude and altitude of position from GPS.

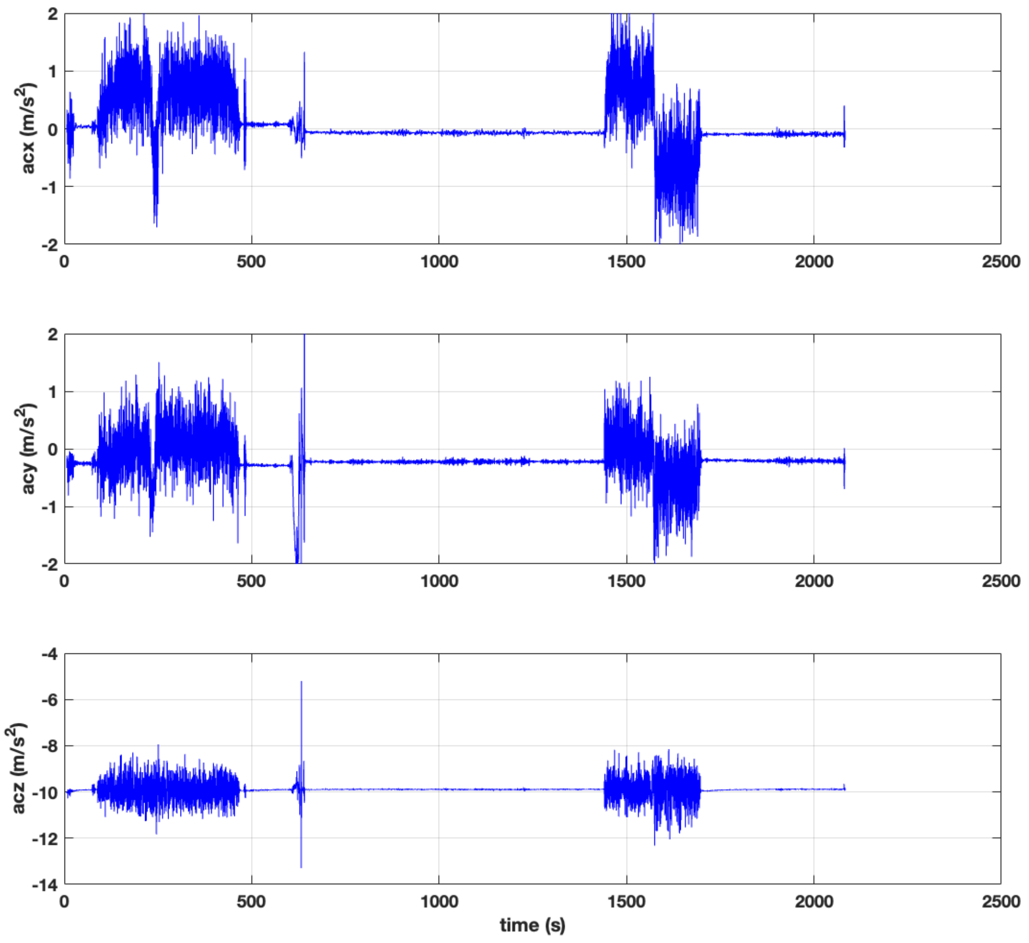

Figure 2: Acceleration

Then, I transformed ECEF coordinates to local north-east-down frame(NED). The required t_min was at 840 s, so I took this position as the origin of NED frame. $(X_{ecef_0}, Y_{ecef_0}, Z_{ecef_0})$ is the NED origin in ECEF frame and $(\varphi_0, \lambda_0, h_0)$ is the GPS position of NED origin.

$$\begin{bmatrix} X_{ned} \\ Y_{ned} \\ Z_{ned} \end{bmatrix} = R^T \left(\begin{bmatrix} X_{ecef} \\ Y_{ecef} \\ Z_{ecef} \end{bmatrix} - \begin{bmatrix} X_{ecef_0} \\ Y_{ecef_0} \\ Z_{ecef_0} \end{bmatrix} \right) \tag{4}$$

$$R = \begin{bmatrix} -\sin\varphi_0 \cos\lambda_0 & -\sin\lambda_0 & -\cos\varphi_0 \cos\lambda_0 \\ -\sin\varphi_0 \sin\lambda_0 & \cos\lambda_0 & -\cos\varphi_0 \sin\lambda_0 \\ \cos\varphi_0 & 0 & -\sin\varphi_0 \end{bmatrix} \tag{5}$$

With all equation above, I could transform GPS position to NED frame. The detailed Matlab code is as followed.

```
%% Convert GPS raw measurements to local NED position values
a = 6378137.0;
f = 1/298.257223563;
b = (1 - f) * a;
t_min = 840; t_max = 1996;
```

```
t_min_index = 4099; t_max_index = 9804;
lat_rad = deg2rad(lat); lon_rad = deg2rad(lon);

% transfrom from GPS to ecef
N_fi = a ^ 2 ./ sqrt((a * cos(lat_rad)).^2 + (b * sin(lat_rad)).^2);
x_ecef = (N_fi + alt) .* cos(lat_rad) .* cos(lon_rad);
y_ecef = (N_fi + alt) .* cos(lat_rad) .* sin(lon_rad);
z_ecef = ((b / a) ^ 2 * N_fi + alt) .* sin(lat_rad);

% set the origin of ned frame
x_ecef_o = x_ecef(t_min_index);
y_ecef_o = y_ecef(t_min_index);
z_ecef_o = z_ecef(t_min_index);
lat_o = lat_rad(t_min_index);
lon_o = lon_rad(t_min_index);

rotation_mat = [
    -sin(lat_o) * cos(lon_o) -sin(lon_o) -cos(lat_o) * cos(lon_o);
    -sin(lat_o) * sin(lon_o) +cos(lon_o) -cos(lat_o) * sin(lon_o);
    +cos(lat_o)              +0          -sin(lat_o)
]';

% transform from ecef to ned
ned = rotation_mat * ([x_ecef'; y_ecef'; z_ecef'] - [x_ecef_o; y_ecef_o; z_ecef_o]);
x_ned = ned(1, :); y_ned = ned(2, :); z_ned = ned(3, :);
```

After transformation, the local NED position plot is shown in figure 3.

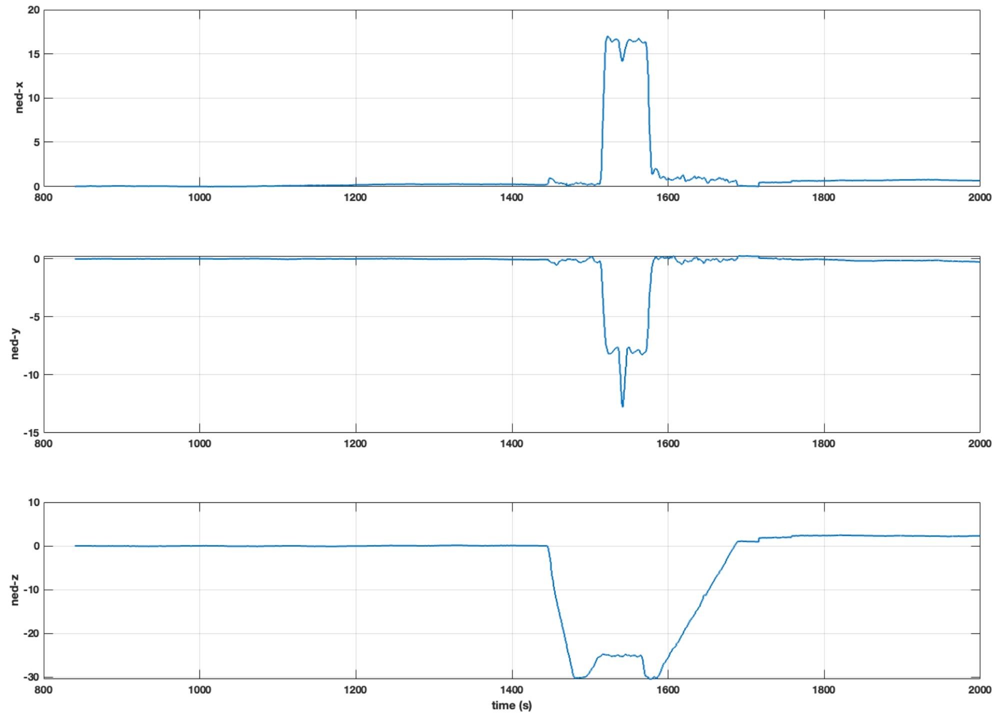

Figure 3: Local NED Position

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

Prediction Step:
First, I calculated the rotation matrix of the UAV body frame relative to the ground frame $R_{g/b}$, where ϕ, θ, φ are yaw, pitch and roll angle respectively.

$$\mathbf{R_{g/b}} = \begin{bmatrix} \cos\phi\cos\theta & \cos\phi\sin\theta\sin\varphi - \sin\phi\cos\varphi & \cos\phi\sin\theta\cos\varphi + \sin\phi\sin\varphi \\ \sin\phi\cos\theta & \sin\phi\sin\theta\sin\varphi + \cos\phi\cos\varphi & \sin\phi\sin\theta\cos\varphi - \cos\phi\sin\varphi \\ -\sin\theta & \cos\theta\sin\varphi & \cos\theta\cos\varphi \end{bmatrix} \tag{6}$$

With the rotation matrix, I could transform the observed acceleration from UAV body frame to NED frame. The position and velocity transition equation could be written as:

$$\mathbf{F_1} = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 & -\frac{dt^2}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 & 0 & -\frac{dt^2}{2} & 0 \\ 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 & -\frac{dt^2}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 & -dt & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -dt & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -dt \end{bmatrix} * \begin{bmatrix} I_6 & 0_3 \\ 0_{3\times 6} & \mathbf{R_{g/b}} \end{bmatrix} \tag{7}$$

$$\mathbf{G_1} = \begin{bmatrix} \frac{dt^2}{2} & 0 & 0 \\ 0 & \frac{dt^2}{2} & 0 \\ 0 & 0 & \frac{dt^2}{2} \\ dt & 0 & 0 \\ 0 & dt & 0 \\ 0 & 0 & dt \end{bmatrix} * \begin{bmatrix} \mathbf{R_{g/b}} & \begin{matrix} 0 \\ 0 \\ g \end{matrix} \end{bmatrix} * \begin{bmatrix} a_x \\ a_y \\ a_z \\ 1 \end{bmatrix}_k \tag{8}$$

$$\begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix}_k = \mathbf{F_1} * \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ b_x \\ b_y \\ b_z \end{bmatrix}_{k-1} + \mathbf{G_1} * \begin{bmatrix} a_x \\ a_y \\ a_z \\ 1 \end{bmatrix}_k \tag{9}$$

And the bias was regarded as a constant, so the acceleration bias transistion equation could be written as the equation below, where $\mathbf{F_2}$ and $\mathbf{G_2}$ are 3×9 zero matrix and 3×4 zero matrix respectively.

$$\begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}_k = \mathbf{F_2} * \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ b_x \\ b_y \\ b_z \end{bmatrix}_{k-1} + \mathbf{G_2} * \begin{bmatrix} a_x \\ a_y \\ a_z \\ 1 \end{bmatrix}_k \tag{10}$$

With equation above, the complete state transition matrix is $\mathbf{F} = [\mathbf{F_1}; \mathbf{F_2}]$ and the complete input control matrix is $\mathbf{G} = [\mathbf{G_1}; \mathbf{G_2}]$. Also, the covariance matrix of state variable should transit as well, where $\mathbf{Q}$ is the covariance matrix of process noise.

$$\mathbf{P_k} = \mathbf{F} * \mathbf{P_{k-1}} * \mathbf{F^T} + \mathbf{Q} \tag{11}$$

Correction Step:
Measurement contains position and velocity, so prediction equation could be written as below, where $\mathbf{H}$ is the measurement matrix:

$$\hat{\mathbf{y}_\mathbf{k}} = \mathbf{H} * \mathbf{x_k} = \begin{bmatrix} I_3 & 0_3 & 0_3 \\ 0_3 & I_3 & 0_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ b_x \\ b_y \\ b_z \end{bmatrix} \tag{12}$$

Then calculate the residual covariance matrix $\mathbf{S}$, which is the difference between the measurement and prediction, and the Kalman gain matrix $\mathbf{W}$, where $\mathbf{R}$ is the covariance matrix of the measurement noise. The vector $\mathbf{y_k}$ comes from the GPS measurement, where the (x, y, z) position is in the NED frame, and (v_x, v_y, v_z) is calculated as the average velocity between GPS signal update intervals.

$$\begin{aligned} \mathbf{S} &= \mathbf{H} * \mathbf{P_k} * \mathbf{H}^\mathbf{T} + \mathbf{R} \\ \mathbf{W} &= \mathbf{P_k} * \mathbf{H} * \mathbf{S}^{-1} \end{aligned} \tag{13}$$

$$\Delta\mathbf{x} = \mathbf{W} * (\mathbf{y_k} - \hat{\mathbf{y}_\mathbf{k}}) \tag{14}$$

Updating Step
According to the correction value, calculate the posterior state variable and the covariance matrix.

$$\mathbf{x_k} = \mathbf{x_k} + \Delta\mathbf{x} \tag{15}$$

$$\mathbf{P_k} = \mathbf{P_k} - \mathbf{W} * \mathbf{S} * \mathbf{W}^\mathbf{T} \tag{16}$$

The corresponding matlab code is as below.

```
%% Implement EKF to estimate NED position and velocity

% initialize EKF matrix
ekf_output = zeros(t_max_index - t_min_index + 1, 9);
eph_mean = 1; epv_mean = 1; dt_mean = 1.0;
var_x = eph_mean / 2; var_y = eph_mean / 2; var_z = epv_mean;

% state variable and cov matrix initialization
x_k = zeros(9, 1);
p_k = diag([var_x, var_y, var_z, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0]);

% measurement matrix
H = eye(6, 9);

% cov mat of measurement noise
var_xyz = diag([var_x, var_y, var_z]);
var_v = diag([2 * var_x / dt_mean ^ 2,
              2 * var_y / dt_mean ^ 2,
              2 * var_z / dt_mean ^ 2]);
cov_xyz_v = diag([var_x / dt_mean, var_y / dt_mean, var_z / dt_mean]);
R = [var_xyz cov_xyz_v; cov_xyz_v var_v];

% check update condition
```

```
waiting_time = 0.0;
last_update_ned = [x_ned(t_min_index); y_ned(t_min_index); z_ned(t_min_index)];

for k = t_min_index + 1 : t_max_index
    dt = t(k) - t(k - 1);
    u_k = [acx(k) acy(k) acz(k) 1]';
    R_ned_body = rotz(psi(k)) * roty(tht(k)) * rotx(phi(k));
    roll_k = phi(k); pitch_k = tht(k); yaw_k = psi(k);

    % calculate F
    trans_f = [eye(3, 3)     dt .* eye(3)    -dt ^ 2 / 2 .* eye(3);
               zeros(3, 3)   eye(3)          -dt .* eye(3)];
    rot_f   = [eye(6)        zeros(6, 3);
               zeros(3, 6)   R_ned_body];
    F = [trans_f * rot_f;
         zeros(3, 6), eye(3)];
    jacobin_f_x = F;

    % calculate G
    trans_g = [dt ^ 2 / 2 * eye(3); dt * eye(3); zeros(3, 3)];
    rot_g = [R_ned_body [0 0 9.8]'];
    G = trans_g * rot_g;

    % cov mat of observation noise
    Q = G * acc_noise * G';

    % prediction step
    x_k = F * x_k + G * u_k;
    p_k = jacobin_f_x * p_k * jacobin_f_x' + Q;
    ekf_output(k - t_min_index + 1, :) = x_k;

    % check new gps signal availablity
    ned_k = [x_ned(k), y_ned(k), z_ned(k)]';
    ned_k_1 = [x_ned(k - 1), y_ned(k - 1), z_ned(k - 1)]';

    waiting_time = waiting_time + dt;
    if isequal(ned_k, ned_k_1)
        new_gps_flag = waiting_time >= 1.0;
    else
        new_gps_flag = true;
    end

    if new_gps_flag
        % correction step
        S = H * p_k * H' + R;
        W = p_k * H' / S;
        y_k = H * x_k;
        delta_y = [ned_k - y_k(1:3) ;
                   (ned_k - last_update_ned) ./ waiting_time - y_k(4:end)];

        % update step
        x_k = x_k + W * delta_y ;
        p_k = p_k - W * S * W';

        waiting_time = 0.0;
        last_update_ned = ned_k;
    end
end
disp("Karman Filter Done!");
```

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

system they are defined in.

State space can be expressed as $(x, y, z, v_x, v_y, v_z, b_x, b_y, b_z)$, where (x, y, z) is the position in NED frame, (v_x, v_y, v_z) is the linear velocity in NED frame and (b_x, b_y, b_z) is the bias of acceleration in the UAV body frame.

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

I extracted a fine-grained portion from the acceleration curve and GPS curve, shown in Figure 4 and 5. It can be observed that the IMU updates at every timestep, while the GPS position only changes every few timesteps(0.8s-1.0s).

Therefore, in my code, I implemented by calculating measurement correction and updating state variable only when new GPS signal arrived. Here is part of my code to determine if there is a new GPS signal.

```
    ned_k = [x_ned(k), y_ned(k), z_ned(k)]';
    ned_k_1 = [x_ned(k - 1), y_ned(k - 1), z_ned(k - 1)]';

    waiting_time = waiting_time + dt;
    if isequal(ned_k, ned_k_1)
        new_gps_flag = waiting_time >= 1.0;
    else
        new_gps_flag = true;
    end
    if new_gps_flag:
        % Karman Filter Correction Step and Update Step
        ...
        waiting_time = 0.0;
```

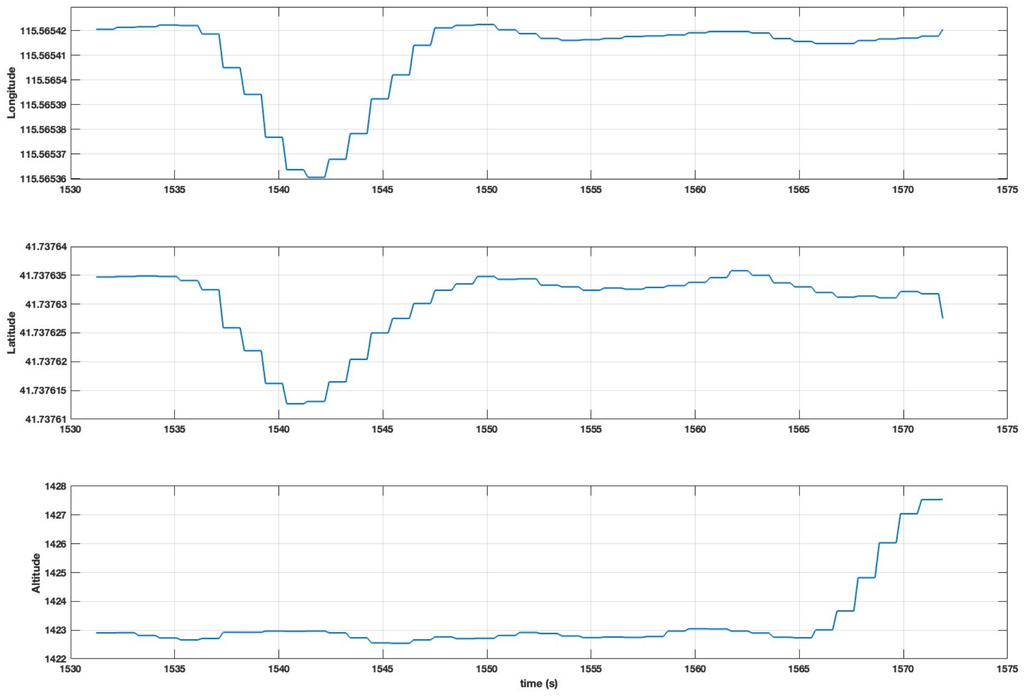

Figure 4: GPS Update Frequency

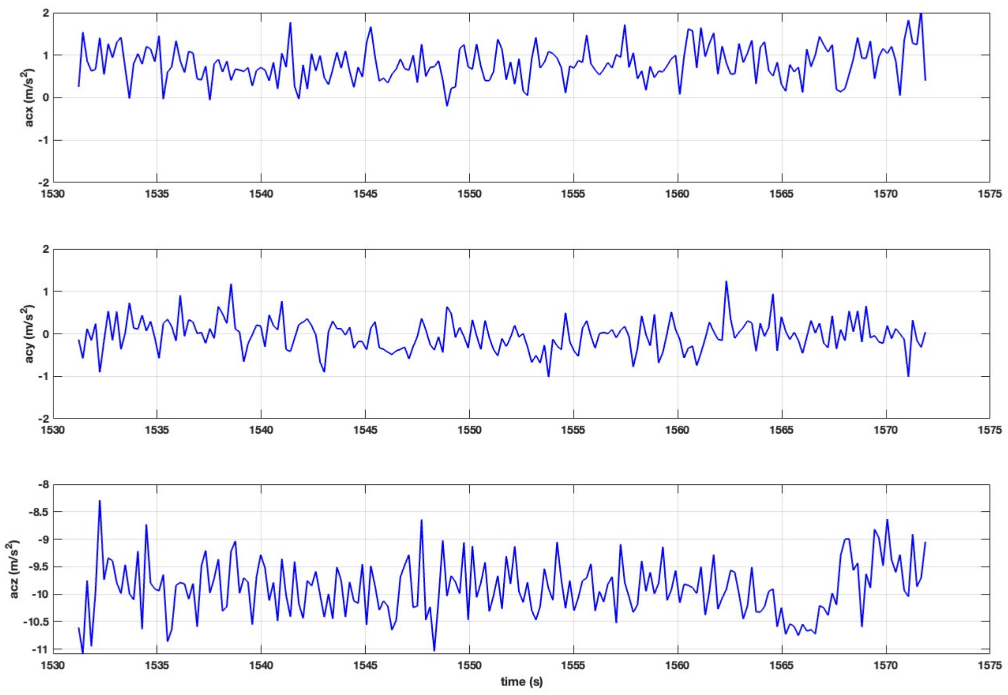

Figure 5: Acceleration Update Frequency

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

Since according to the installation requirements, the EKF should start 10 minutes before the UAV takes off, so initially, the UAV is located at the origin of the NED coordinate system, with a velocity of 0, and the acceleration bias is assumed to be 0. So the initial state variable could be written as:

$$\mathbf{x_0} = (x, y, z, v_x, v_y, v_z, b_x, b_y, b_z) = zeors(9, 1) \tag{17}$$

The GPS measurements are noisy. According to the problem requirements, the noise in the horizontal direction is eph, and in the vertical direction is epv. Therefore, based on the properties of the NED coordinate system and eph/epv, I can calculate the coordinate variance of the measurements in the NED frame as the equation below:

$$\begin{gathered} var(z) = epv = 1 \\ var(x) = var(y) = \frac{eph}{2} = \frac{1}{2} \end{gathered} \tag{18}$$

Initially, the UAV is clearly stationary, so the velocity variance is equal to 0. For the acceleration bias, I initialized the variance to a value of 1. Additionally, I assumed all the variables were independent at the initial state, so the covariance matrix can be written as a diagonal matrix:

$$\mathbf{p_0} = diag([\frac{1}{2}, \frac{1}{2}, 1, 0, 0, 0, 1, 1, 1]); \tag{19}$$

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

For the $\mathbf{R}$ matrix, it is actually the covariance matrix of measurement noise. The measurement variance of position has been calculated in question 7. For the velocity, I calculated it using the equation below, where k is the current step and j is the previous GPS update step. Δt is the GPS update period, which is approximately 1 second.

$$\begin{aligned} var(v_x) &= var[(x_k - x_j)/\Delta t] = \frac{2}{\Delta t^2} var(x_k) = 1 \\ var(v_y) &= var[(y_k - y_j)/\Delta t] = \frac{2}{\Delta t^2} var(y_k) = 1 \\ var(v_z) &= var[(z_k - z_j)/\Delta t] = \frac{2}{\Delta t^2} var(z_k) = 2 \end{aligned} \tag{20}$$

Additionally, since my measured velocity is the average velocity calculated from GPS positions, the covariance between velocity and position cannot be ignored.

$$\begin{aligned} cov(x, v_x) &= cov(x_k, \frac{x_k - x_i}{\Delta t}) \\ &= \frac{1}{\Delta t} cov(x_k, x_k - x_i) \\ &= \frac{1}{\Delta t}(cov(x_k, x_k) - cov(x_k, x_i)) \\ &= \frac{1}{\Delta t} var(x) = \frac{1}{2} \\ cov(y, v_y) &= \frac{1}{\Delta t} var(y) = \frac{1}{2} \\ cov(z, v_z) &= \frac{1}{\Delta t} var(z) = 1 \end{aligned} \tag{21}$$

In conclusion, the R matrix could be written as:

$$\mathbf{R} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 \end{bmatrix} \tag{22}$$

For Q matrix, it is actually the covariance matrix of observation noise, which could be written as formula below, where q_x, q_y, q_z are the variance of acceleration.

$$\mathbf{Q} = \mathbf{G} * \begin{bmatrix} q_x & 0 & 0 & 0 \\ 0 & q_y & 0 & 0 \\ 0 & 0 & q_z & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} * \mathbf{G}^{\mathrm{T}} \tag{23}$$

In my assignment, I tested three different sets of acceleration noise variance: small noise corresponds to $(q_x, q_y, q_z) = (0.1, 0.1, 0.1)$, medium noise corresponds to $(q_x, q_y, q_z) = (1, 1, 1)$, and large noise corresponds to $(q_x, q_y, q_z) = (5, 5, 5)$.

The velocity and position curves output by the Kalman filter are shown in the following figure. The red line represents the GPS measurements, the green line represents the predicted values with small acceleration variance, the blue line represents the predicted values with medium acceleration variance, and the yellow line represents the predicted values with large acceleration variance.

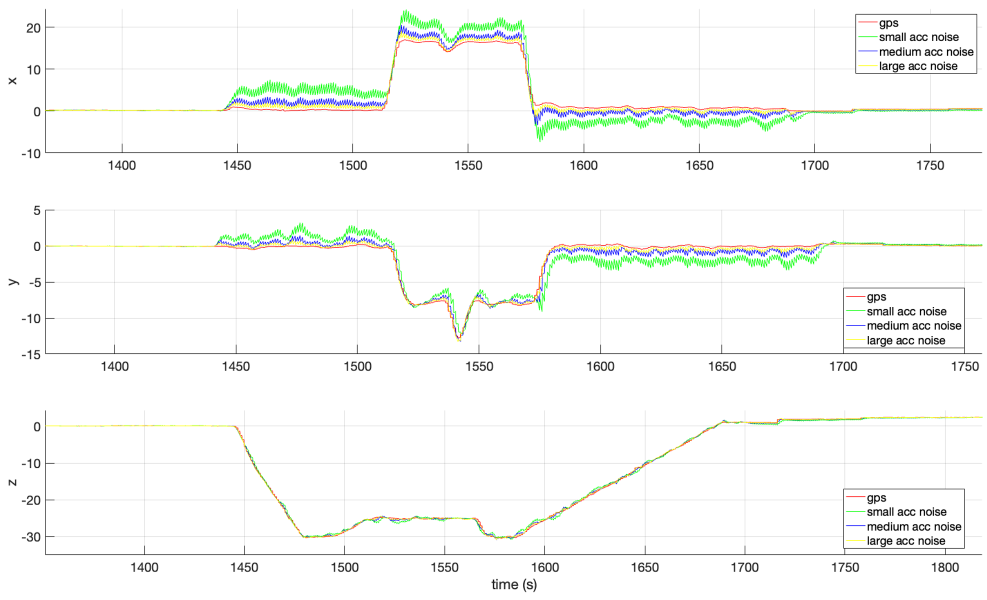

Figure 6: Position Curve

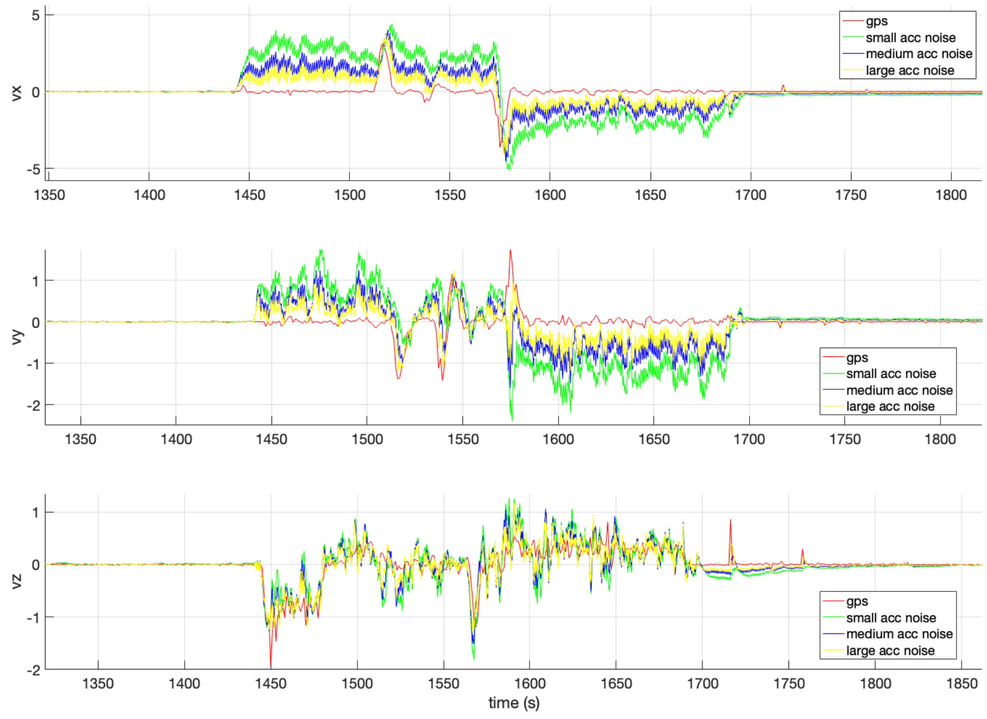

Figure 7: Velocity Curve

From these two curves, it can be seen that the Kalman filter balances between observation noise and measurement noise. When the observation noise is small, the system trusts the predicted values from the state transition more; when the observation noise is large, the system relies more on the measurements. Therefore, in the figure, curves with larger acceleration noise are closer to the GPS measurement curve.

9. Have the accelerometer bias values converged in the end? What values did they converge on?
Yes, the acceleration bias ultimately converged. For different levels of acceleration noise, the biases are generally similar. As shown in the figure, the three plots are nearly coincident. The x bias is approximately -0.03, the y bias is approximately -0.2, and the z bias is approximately -0.09.

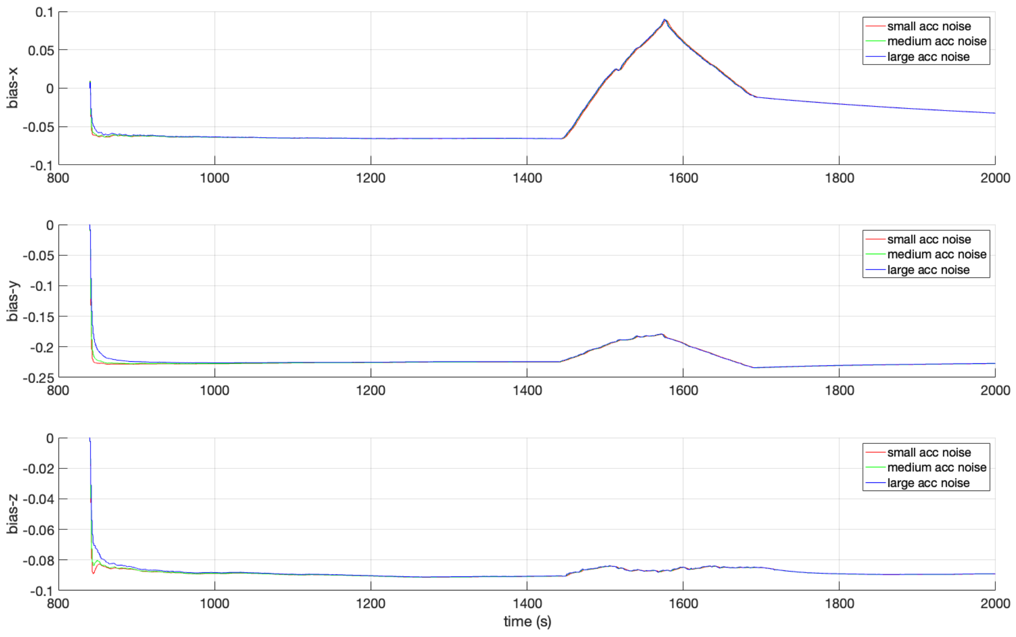

Figure 8: Acceleration Bias

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?
I will additionally include a drift rate d in the state variables. So in my karman filter system, state variables could be written as $(x, y, z, v_x, v_y, v_z, b_x, b_y, b_z, d)$. Acceleration bias and drift would transit like equation below:

$$\begin{aligned} b_x(k+1) &= b_x(k) + d(k) * \Delta t \\ b_y(k+1) &= b_y(k) + d(k) * \Delta t \\ b_z(k+1) &= b_z(k) + d(k) * \Delta t \\ d(k+1) &= d(k) \end{aligned} \tag{24}$$

I initialized d_0 to -1, with a variance of 1. However, after a few update iterations, the value of d quickly became 0, as shown in the figure 9. And the final acceleration bias plot is similar to the constant bias situation, as shown in figure 10.

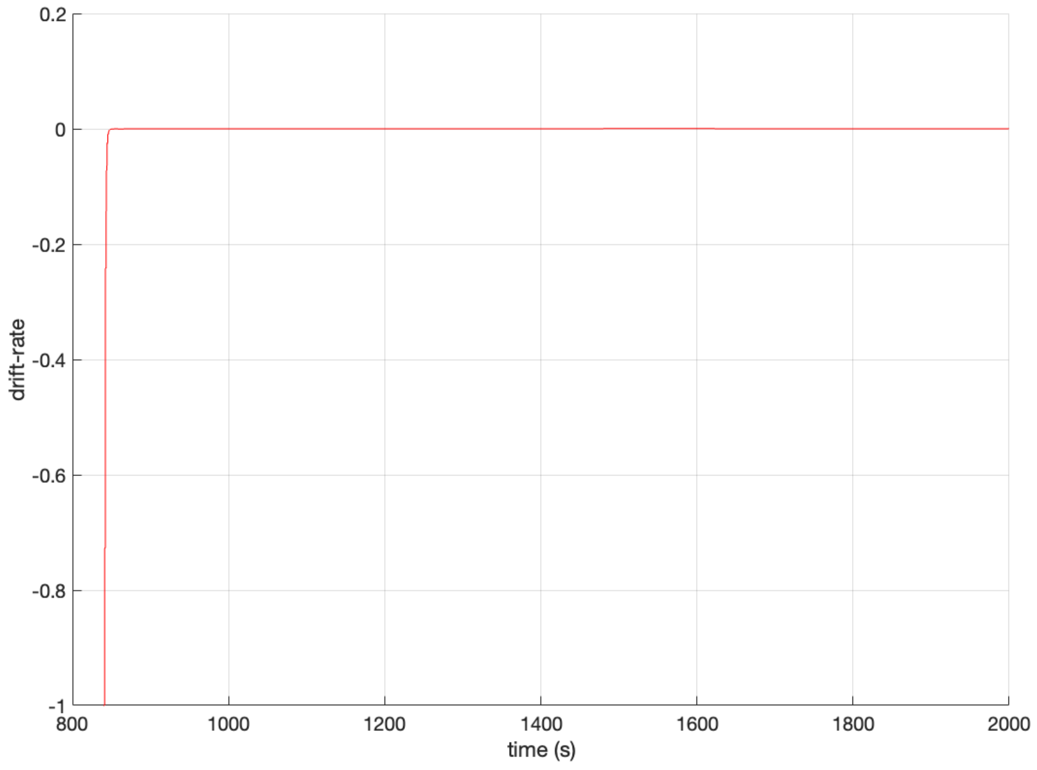

Figure 9: Drift Rate

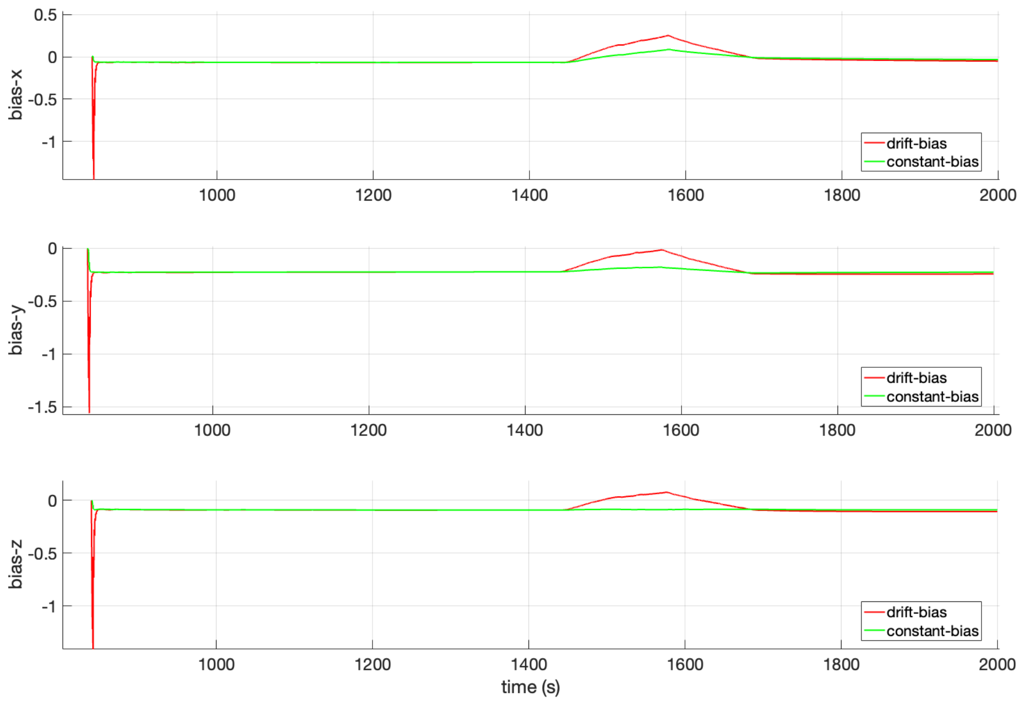

Figure 10: Drift Bias