Project 1

Robot Localization and Navigation

Arjun Raja – ar6841

1. Project overview

This project entails the design of an Extended Kalman Filter to estimate the state of a Micro Aerial Vehicle. The state of the MAV contains position, velocity, and orientation, and gyroscope sensor bias and accelerometer sensor bias. The body frame acceleration and angular velocity from the on board IMU are being used as the inputs. The process and measurement noise follow normal distributions.

2. Process model

The state of the MAV is a vector X:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \\ \mathbf{x}_5 \end{bmatrix} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \\ \dot{\mathbf{p}} \\ \mathbf{b}_g \\ \mathbf{b}_a \end{bmatrix} = \begin{bmatrix} \text{position} \\ \text{orientation} \\ \text{linear velocity} \\ \text{gyroscope bias} \\ \text{accelerometer bias} \end{bmatrix} \in \mathbf{R}^{15}$$

In MATLAB this can be represented by a 15x1 vector, with the vectors : $\mathbf{x}_1 = [x, y, z]^\top$, $\mathbf{x}_2 = [angle\_x, angle\_y, angle\_z]^\top$, $\mathbf{x}_3 = [vx, vy, vz]^\top$, $\mathbf{x}_4 = [bg1, bg2, bg3]^\top$, $\mathbf{x}_5 = [ba1, ba2, ba3]^\top$. The data for the state of the MAV is given in the code by uPrev(i , 1).

2.1 R matrix

The rotation order is ZYX by the Euler angles $[angle\_x, angle\_y, angle\_z] = [\phi, \theta, \psi]$ in this project. So we can find the rotation matrix R

$$^{W}R_B = R_z(\psi)R_y(\theta)R_x(\phi)$$

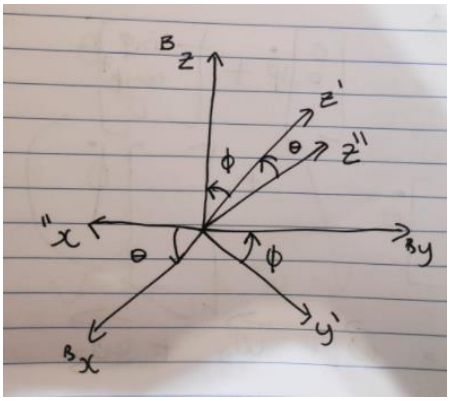
R is precalculated and implemented in the code, thus the required values can be substituted in.

2.2 G matrix calculation

The G matrix is a mapping from Euler angle rates to angular velocity. Our input is the **body frame angular velocity** of the robot from the gyroscope, and we need to map it to the Euler angle rates so we can find $\dot{\boldsymbol{x}}_{\mathbf{2}}$**.**

$$\dot{x}_2 = G^{-1} \quad ^{B}\omega$$

The G matrix can be found by splitting the angular velocity into the angular velocity contributions of the Euler angle rates. In other words, angular velocity contribution is the (Euler angle rate) ×(Projection of axis of rotation in the body frame).



$$^{B}\omega = \begin{bmatrix}1\\0\\0\end{bmatrix}\dot{\phi} + \begin{bmatrix}0\\\cos\phi\\-\sin\phi\end{bmatrix}\dot{\theta} + \begin{bmatrix}-\sin\theta\\\cos\theta\sin\phi\\\cos\theta\cos\phi\end{bmatrix}\dot{\psi}$$

$$^{B}\omega = \begin{bmatrix}1 & 0 & -\sin\theta\\0 & \cos\phi & \cos\theta\sin\phi\\0 & -\sin\phi & \cos\theta\cos\phi\end{bmatrix}\begin{bmatrix}\dot{\phi}\\\dot{\theta}\\\dot{\psi}\end{bmatrix}$$

$$^{B}\omega = G\dot{x}_2$$

As the projected the axes of rotation are in the body frame, the angular velocity in the derivation is in the body frame. There is no need to rotate the angular velocity, so there's no need to multiply G with Rinv.

As G^{-1} is used in the process model it has been pre calculated and only the Euler angle values need to be substituted to find G^{-1}

Code Sample:

```
%G=[1 0 -sin(angle_y);0 cos(angle_x) sin(angle_x)*cos(angle_y);0 -sin(angle_x) cos(angle_x)*cos(angle_y)];
%simplify(inv(G))
%%precomputing Ginv
Ginv=[1, (sin(angle_x)*sin(angle_y))/cos(angle_y), (cos(angle_x)*sin(angle_y))/cos(angle_y);0, cos(angle_x), -sin(angle_x);0, sin(angle_x)/cos(angle_y),
```

2.3 Process model

The process model assumes that $\dot{x} = f(x, u, n)$

$$\dot{\mathbf{x}} = \begin{bmatrix}\mathbf{x}_3\\G(\mathbf{x}_2)^{-1}\left(\boldsymbol{\omega_m} - \mathbf{x}_4 - \mathbf{n}_g\right)\\\mathbf{g} + R(\mathbf{x}_2)(\mathbf{a}_m - \mathbf{x}_5 - \mathbf{n}_a)\\\mathbf{n}_{bg}\\\mathbf{n}_{ba}\end{bmatrix} = f(x, u, n)$$

Where $\mathbf{g}$ is acceleration due to gravity implemented by the vector $[0,0,-9.81]^{\top}$, $\mathbf{n}_{bg}$is the noise due to gyroscope bias and $\mathbf{n}_{ba}$is noise due to accelerometer bias. The inputs to the system (angular velocity and acceleration) are represented by the vector U, and the process noise is represented by the noise vector N. $\mathbf{x}$ is calculated and implemented in the prediction step around the point $(\mu_{t-1}u_t, 0)$.

Code Sample:

```
%% Process model (values are subbed in)
% Substitute u(t-1) and input values in f(u(t-1),input,0)

g=[0;0;-9.81]; %m/s
q_dot=Ginv*(angVel-uPrev(10:12,1));
body_acc=g+R*(acc-uPrev(13:15,1));

state_dot=vertcat(uPrev(7:9,1),q_dot,body_acc,zeros(6,1)); %x_dot
```

```
48    % X=[x; y; z; anglex; angley; anglez; vx; vy; vz; bg1; bg2; bg3; ba1; ba2; ba3]
49    % U=[W1; W2; W3; A1; A2; A3];
50    % N=[nv1; nv2; nv3; ng1; ng2; ng3; na1; na2; na3; nbg1; nbg2; nbg3; nba1; nba2; nba3]
```

The process noise has a covariance matrix Q [15x15] and is randomly assigned when the program is run using the MATLAB rand() and diag() functions. It remains constant between iterations so Q was implemented by persistent data type.

3. Measurement model

The measurement model can be represented by the equation,

$$Z_t = g(x, v_t) = C_t x_t + v_t$$
$$v_t \sim N(0, R_t)$$

The measurement model is linear.

Where v_t is the measurement noise. R_t [6x6] and is randomly assigned when the program is run using the MATLAB rand() and diag() functions. It remains constant between iterations so R_t was implemented by persistent data type.

3.1 Part 1

The project is split into two parts, in part one we are measuring the position $\mathbf{x}_1$ and orientation $\mathbf{x}_2$.

C_t in part one is a [6x15] matrix $\begin{bmatrix} I_3 & 0 & 0 & 0 & 0 \\ 0 & I_3 & 0 & 0 & 0 \end{bmatrix}$ where I_3 is a 3x3 identity matrix

```
5     Ct = [eye(3), zeros(3,12);zeros(3,3), eye(3), zeros(3,9)];
6     %Wt not needed as measurement model is linear
```

So $g(x, v_t)$ gives us a 6x1 vector

3.2 Part 2

In part two we measure velocity $\mathbf{x}_3$ and C_t in part two is a [3x15] matrix $[0 \quad I_3 \quad 0 \quad 0 \quad 0]$

```
5     Ct = [zeros(3,6),eye(3),zeros(3,6)]; %this changed from part1
6     %Wt not needed as measurement model is linear
```

Here $g(x, v_t)$ gives us a 3x1 vector

4. Prediction step

As the process is non-linear, the Extended Kalman Filter linearizes the system around $\dot{\mathbf{x}}(\mu_{t-1}u_t, 0)$.

And the discrete prediction step can be found to be

$$\bar{\mu}_t = \mu_{t-1} + \delta t f(\mu_{t-1}, u_t, 0)$$
$$\bar{\Sigma}_t = F_t \Sigma_{t-1} F_t^T + V_t Q_d V_t^T$$

$$\dot{x} = f(x, u, n)$$
$$n \sim N(0, Q)$$
$$A_t = \left.\frac{\partial f}{\partial x}\right|_{\mu_{t-1}, u_t, 0}$$
$$U_t = \left.\frac{\partial f}{\partial n}\right|_{\mu_{t-1}, u_t, 0}$$
$$F_t = I_{15} + \delta t A_t$$
$$V_t = U_t$$
$$Q_d = Q\delta t$$

The step size dt = $\delta t$is given as an input into the pred_step function, it's calculation is shown in the main loop description below. The calculation of the process noise has a covariance matrix Q was shown above in the process model.

The Jacobians A_t and U_t were pre calculated and are implemented as persistent symbolic functions using matlabFunction(), this was done to increase the speed of the program.

Code Sample:

```
% W=[W1;W2;W3]
% A=[A1;A2;A3]
% f=vertcat(x3,Ginv*(W-x4-ng),g+R*(A-x5-na),nbg,nba)
% At=jacobian(f,X)
% % Bt=jacobian(f,u)
% Ut=jacobian(f,N)
```

```
persistent SymAt SymUt %sub values into these to find jacobians
if isempty(SymAt)
    syms x y z anglex angley anglez vx vy vz bg1 bg2 bg3 ba1 ba2 ba3 nv1 nv2 nv3 ng1 ng2 ng3 na1 na2 na3 nbg1 nbg2 nbg3 nba1 nba2 nba3 W1 W2 W3 A1 A2 A3
    jacobAt(x, y, z, anglex, angley, anglez, vx, vy, vz, bg1, bg2, bg3, ba1, ba2, ba3, nv1, nv2, nv3, ng1, ng2, ng3, na1, na2, na3, nbg1, nbg2, nbg3, nba1, nba2, nba3
    SymAt=matlabFunction(jacobAt);
    % Can make cleaner by using vertcat()
    jacobUt(x, y, z, anglex, angley, anglez, vx, vy, vz, bg1, bg2, bg3, ba1, ba2, ba3, nv1, nv2, nv3, ng1, ng2, ng3, na1, na2, na3, nbg1, nbg2, nbg3, nba1, nba2, nba3
    SymUt=matlabFunction(jacobUt);
end
% substitutions in At and Ut
%sub_vals=horzcat(transpose(uPrev),zeros(1,15),transpose(angVel),transpose(acc));

%% At Ut and Ft calculation

At=SymAt(uPrev(1,1),uPrev(2,1),uPrev(3,1),uPrev(4,1),uPrev(5,1),uPrev(6,1),uPrev(7,1),uPrev(8,1),uPrev(9,1),uPrev(10,1),uPrev(11,1),uPrev(12,1),uPrev(13,1),uPrev(14,1
%Ut = Vt so im not wasting memory copying it

Ut=SymUt(uPrev(1,1),uPrev(2,1),uPrev(3,1),uPrev(4,1),uPrev(5,1),uPrev(6,1),uPrev(7,1),uPrev(8,1),uPrev(9,1),uPrev(10,1),uPrev(11,1),uPrev(12,1),uPrev(13,1),uPrev(14,1

Ft=eye(15)+dt*At; %Ft calculation
```

The values for the state (form uPrev), noise (all 0 as we linearize around N=0) and inputs (angVel and acc) are substituted directly into the symbolic MATLAB functions to calculate At and Ut. **The prediction step stays the same for part one and part two.**

The pred_step function finally gives us the uEst and covarEst to be used in the update model

Code sample:

```
97    %% Pred_step outputs
98    uEst=uPrev+dt*state_dot;
99    covarEst=Ft*covarPrev*transpose(Ft)+Ut*Qd*transpose(Ut);
```

5. Update step

As the measurement model is linear, the update step takes the form:

$$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$$
$$\Sigma_t = \bar{\Sigma}_t - K_t C_t \bar{\Sigma}_t$$
$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + R_t)^{-1}$$

Where Kt is the Kalman gain and z_t the measurement.

The calculations for R_t and C_t are the same as discussed above in the measurement models

5.1 Part 1

Rt is the covariance matrix of the measurement noise and is a 6x6 diagonal matrix in part one. Ct was the same as found in measurement model part one.

5.2 Part 2

Rt in part two is a 3x3 diagonal matrix. Ct was the same as found in measurement model part two

The update step gives us uCurr and covarCurr.

```
17    Kt=covarEst*transpose(Ct)*inv(Ct*covarEst*transpose(Ct)+Rt);
18    covar_curr=covarEst-Kt*Ct*covarEst;
19    uCurr=uEst+Kt*(z_t-Ct*uEst);
```

6. Main Loop

```
% determine dt
dt=sampledTime(i)-prevTime;
prevTime=sampledTime(i); %for the next loop

% Inputs
angVel=sampledData(i).omg;
acc=sampledData(i).acc;

[covar_est,u_est]=pred_step(uPrev,covarPrev,angVel,acc,dt);
[u_curr,covar_curr]=upd_step(Z(:,i),covar_est,u_est);

uPrev=u_curr;
covarPrev=covar_curr;
savedStates(:,i)=u_curr;
```

In the main loop, first dt is determined by finding the difference between the sampled time and previous sampled time. The angular velocity and acceleration are read from the sampled data structure which contains the gyroscope and accelerometer data.

Then the state estimate (u_est) and covariance estimate (covar_est) are found by feeding the previous state, previous covariance, angular velocity, acceleration and time step dt into the prediction step function (pred_step).

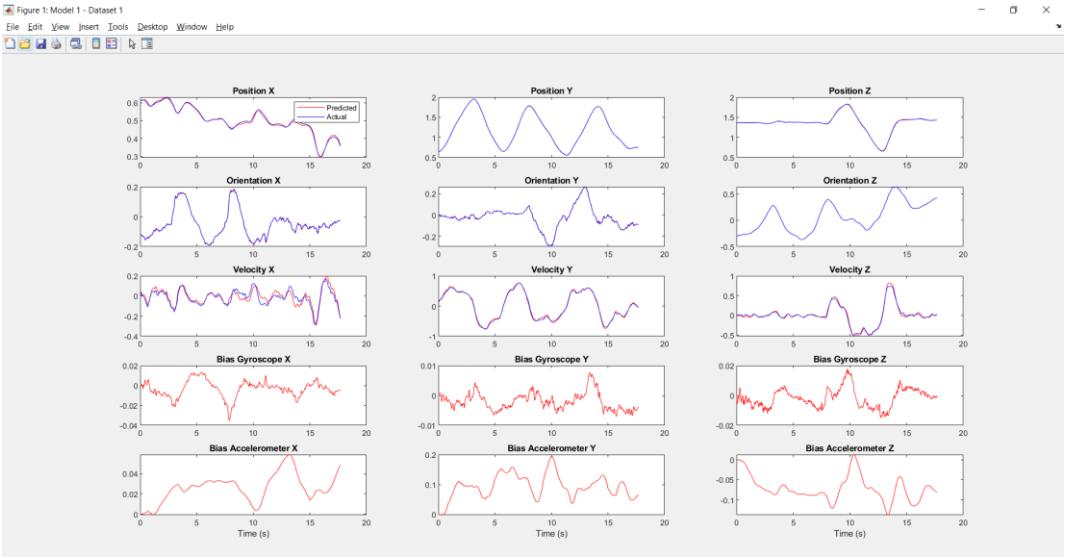
The current measurement, state estimate and covariance estimate are fed into the update step function (upd_step) so we can find the current state(u_curr) and current covariance (cov_curr). The current values, u_curr and covar_curr are saved and used in the next iteration of the loop. All values of u_curr are also saved in the savedStates matrix so it can be plotted later.

7. Results

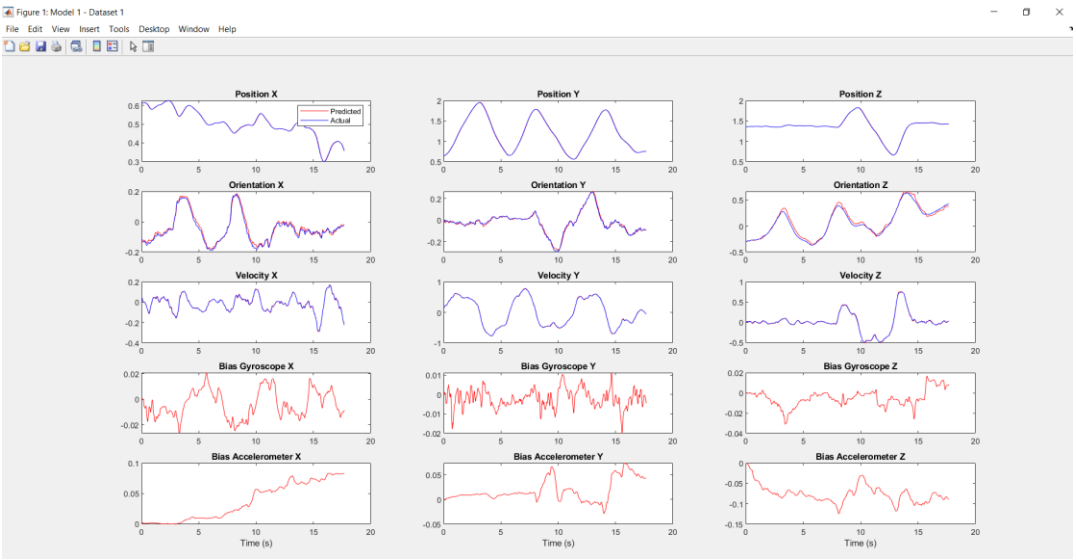
Given below are the plots of the MAV state under different data sets and measurement models.

7.1 Data set 1

Part 1

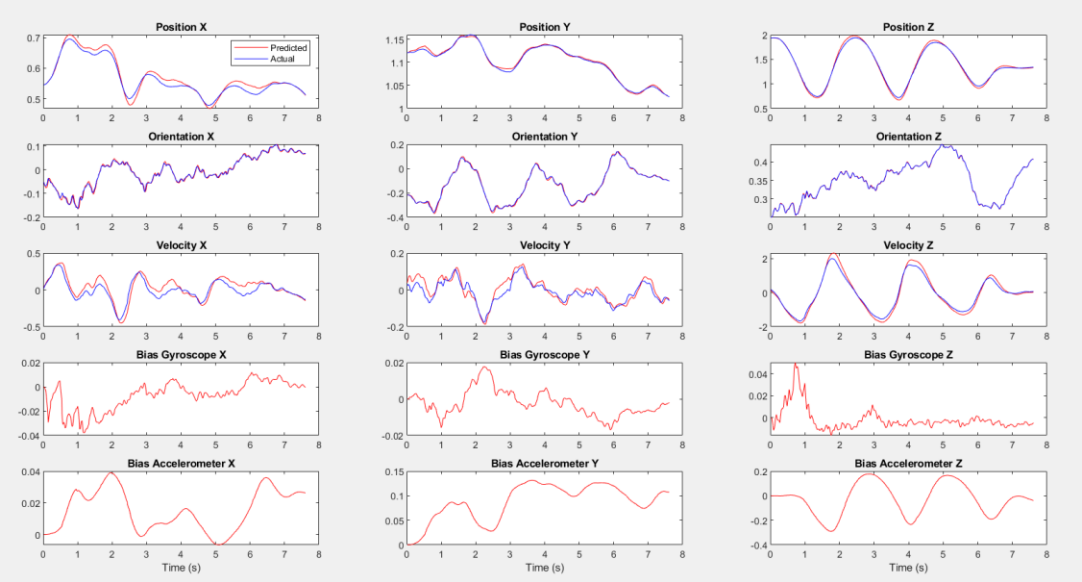


Part 2

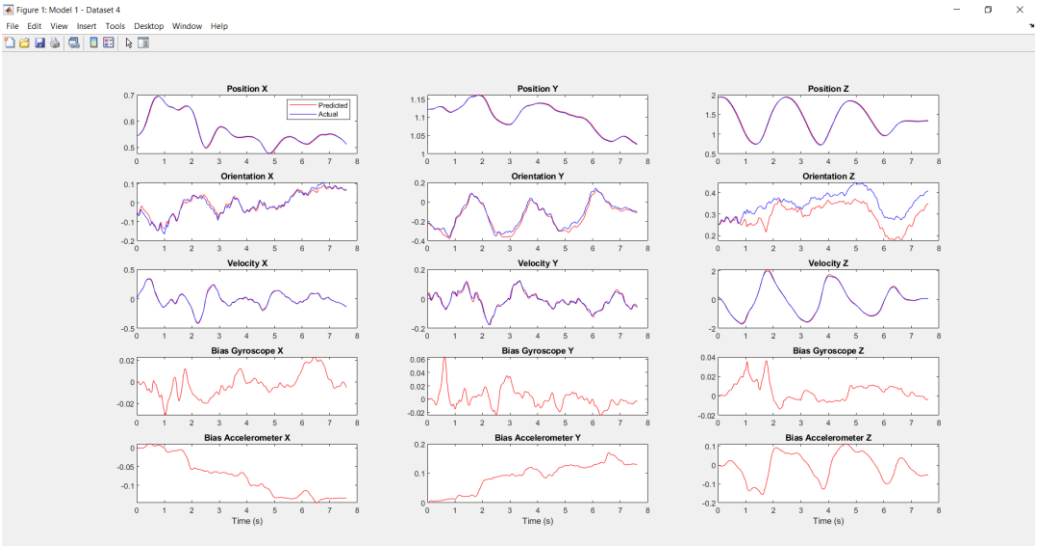


7.2 Data set 4

Part 1

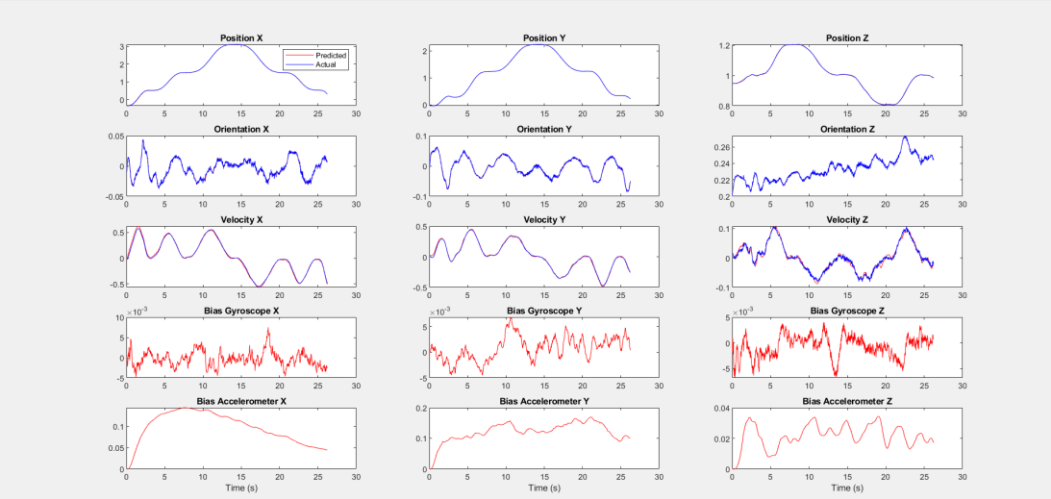


Part 2

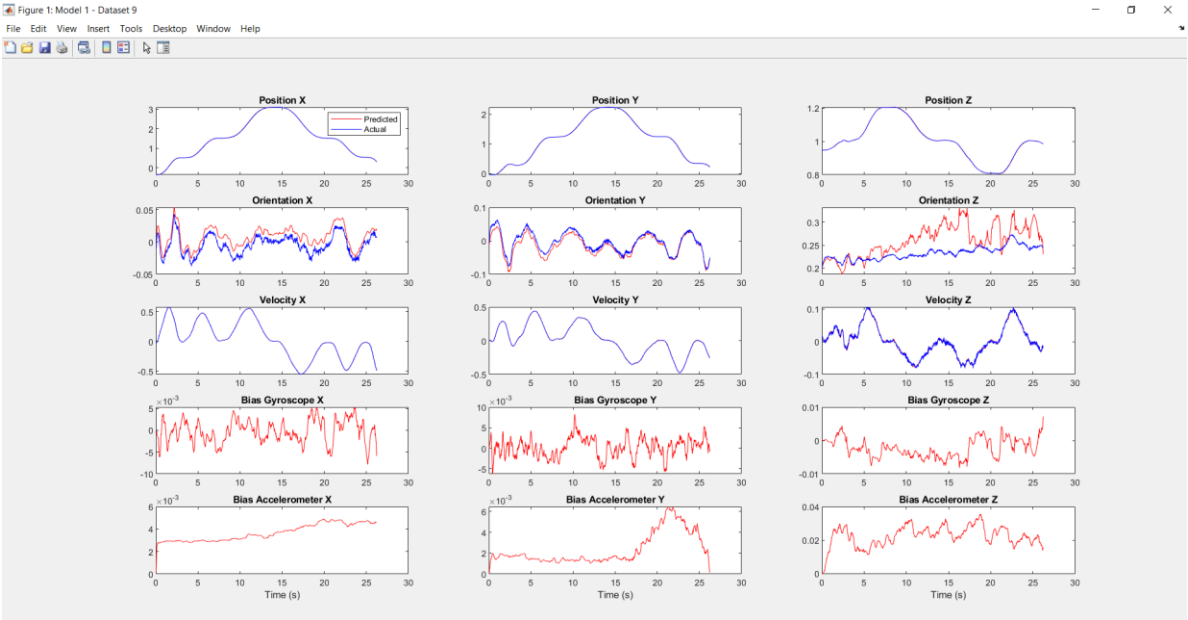


7.3 Data set 9

Part 1



Part 2



8. Discussions

The Extended Kalman filter estimated the state of the MAV with high accuracy. When only the position and orientation are used in the measurement model, as in part one, the extended Kalman filter works well and tracks position, orientation velocity, gyroscope and accelerometer biases . But when the measurement model only measures the velocity of the MAV, as done in part two, the accuracy of the filter drops a bit and there is a drift in the yaw / angle_z. This might be because the filter does not have enough information, we only measured the velocity to calculate the Kalman gain which doesn't correct the yaw enough, which causes its yaw value to drift as the Kalman gain never corrects it completely and the process noise adds up. It's possible to reduce the error in this case by scaling the process noise covariance matrix Q down, but it's not done for the purposes of this report as to make the phenomenon apparent.