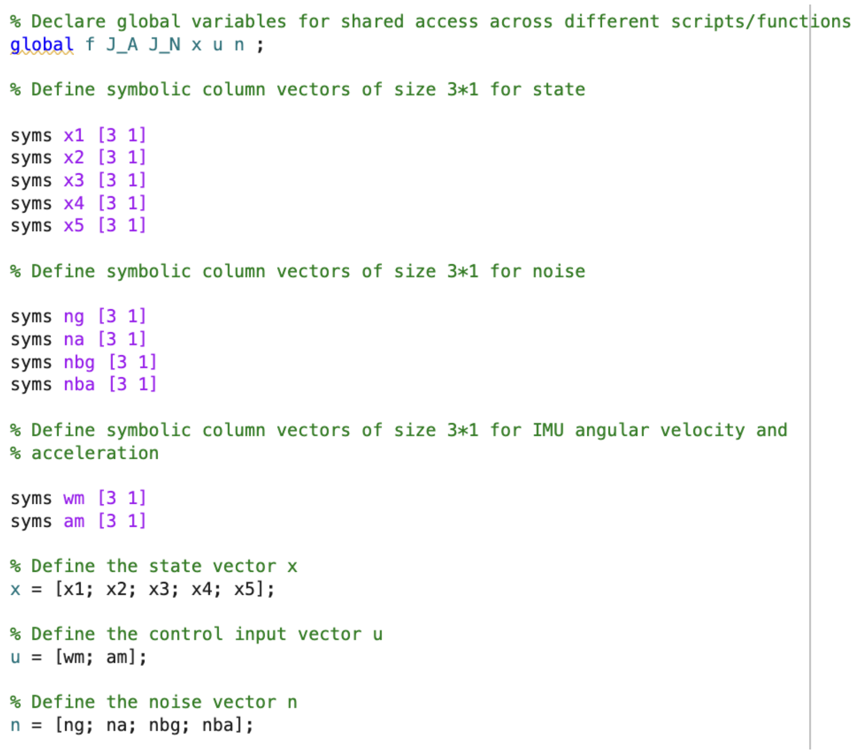
Robot Localisation and Navigation Project 1 Report:

Name: Santosh Srinivas Ravichandran
Net ID: sr6411

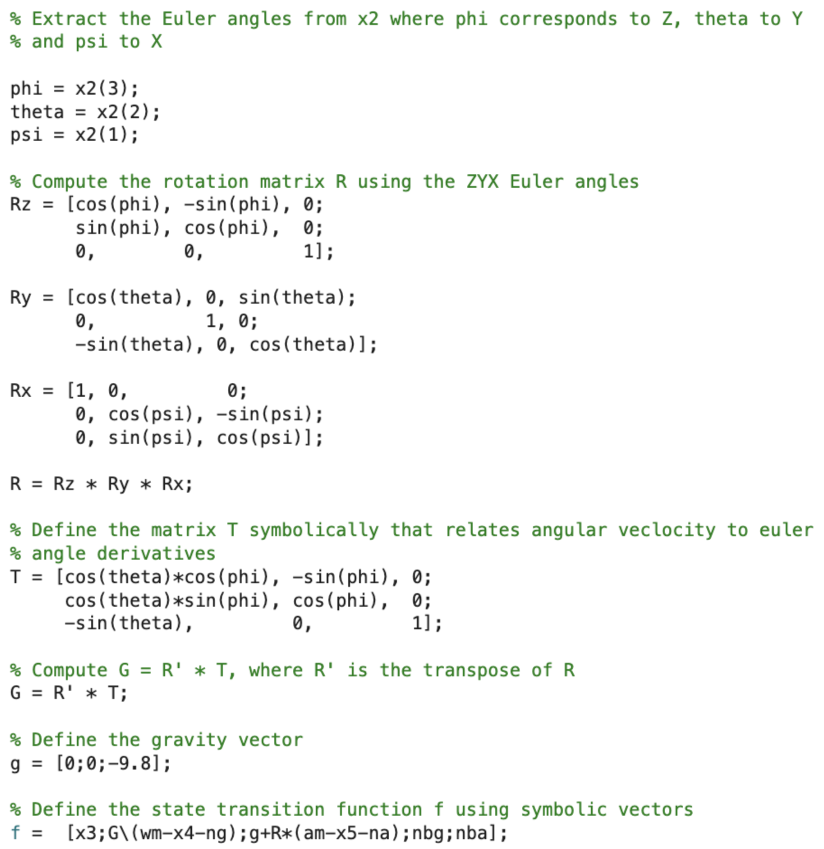
Kalman Filter Part 1:

Once the initialisation script is run, the IMU data and the vicon data is synchronized at the same time.



```
% Declare global variables for shared access across different scripts/functions
global f J_A J_N x u n ;

% Define symbolic column vectors of size 3*1 for state

syms x1 [3 1]
syms x2 [3 1]
syms x3 [3 1]
syms x4 [3 1]
syms x5 [3 1]

% Define symbolic column vectors of size 3*1 for noise

syms ng [3 1]
syms na [3 1]
syms nbg [3 1]
syms nba [3 1]

% Define symbolic column vectors of size 3*1 for IMU angular velocity and
% acceleration

syms wm [3 1]
syms am [3 1]

% Define the state vector x
x = [x1; x2; x3; x4; x5];

% Define the control input vector u
u = [wm; am];

% Define the noise vector n
n = [ng; na; nbg; nba];
```

I define **global variables** corresponding to the state, noise and control input vectors for the sake of reducing time complexity since the variables can be computed only once and be used anywhere in the script. These variables are **symbolic variables** and used to form the function f ie $\dot{x}$; we know that this would be used for computing the jacobian of f only once symbolically, once again reducing time complexity.

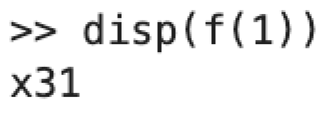


```matlab
% Extract the Euler angles from x2 where phi corresponds to Z, theta to Y
% and psi to X

phi = x2(3);
theta = x2(2);
psi = x2(1);

% Compute the rotation matrix R using the ZYX Euler angles
Rz = [cos(phi), -sin(phi), 0;
      sin(phi), cos(phi),  0;
      0,        0,         1];

Ry = [cos(theta), 0, sin(theta);
      0,          1, 0;
      -sin(theta), 0, cos(theta)];

Rx = [1, 0,         0;
      0, cos(psi), -sin(psi);
      0, sin(psi), cos(psi)];

R = Rz * Ry * Rx;

% Define the matrix T symbolically that relates angular veclocity to euler
% angle derivatives
T = [cos(theta)*cos(phi), -sin(phi), 0;
     cos(theta)*sin(phi), cos(phi),  0;
     -sin(theta),         0,         1];

% Compute G = R' * T, where R' is the transpose of R
G = R' * T;

% Define the gravity vector
g = [0;0;-9.8];

% Define the state transition function f using symbolic vectors
f =  [x3;G\(wm-x4-ng);g+R*(am-x5-na);nbg;nba];
```

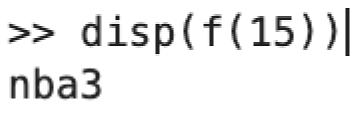
I compute the Rotation Matrix using ZYX Euler angles and define f in terms of state, noise and control input vectors symbolically.

Example:

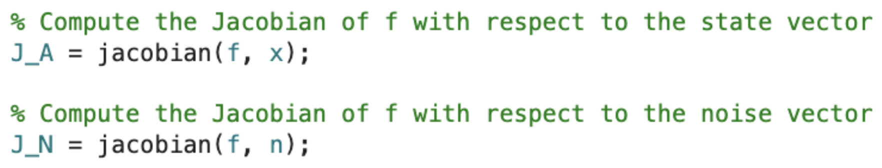
F is now symbolic and its first element is the first element of the x3 vector (linear velocity)



```
>> disp(f(1))
x31
```

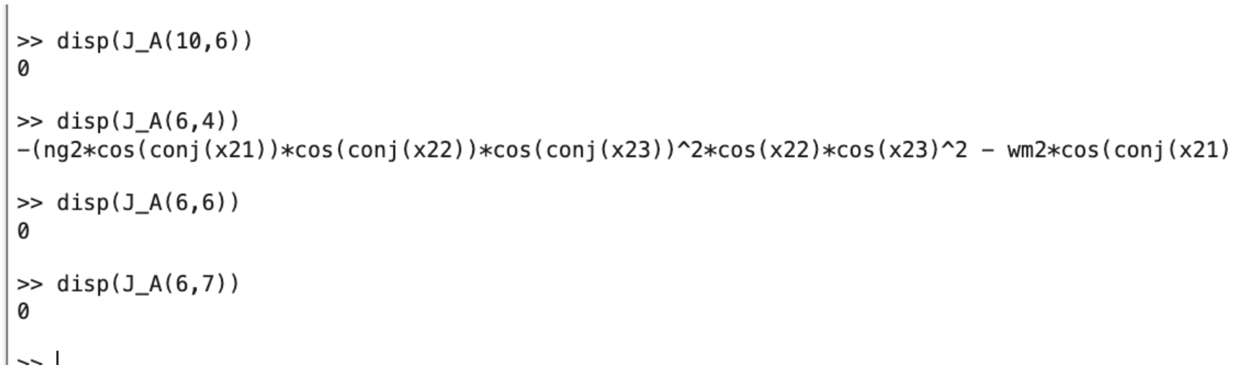


```
>> disp(f(15))
nba3
```

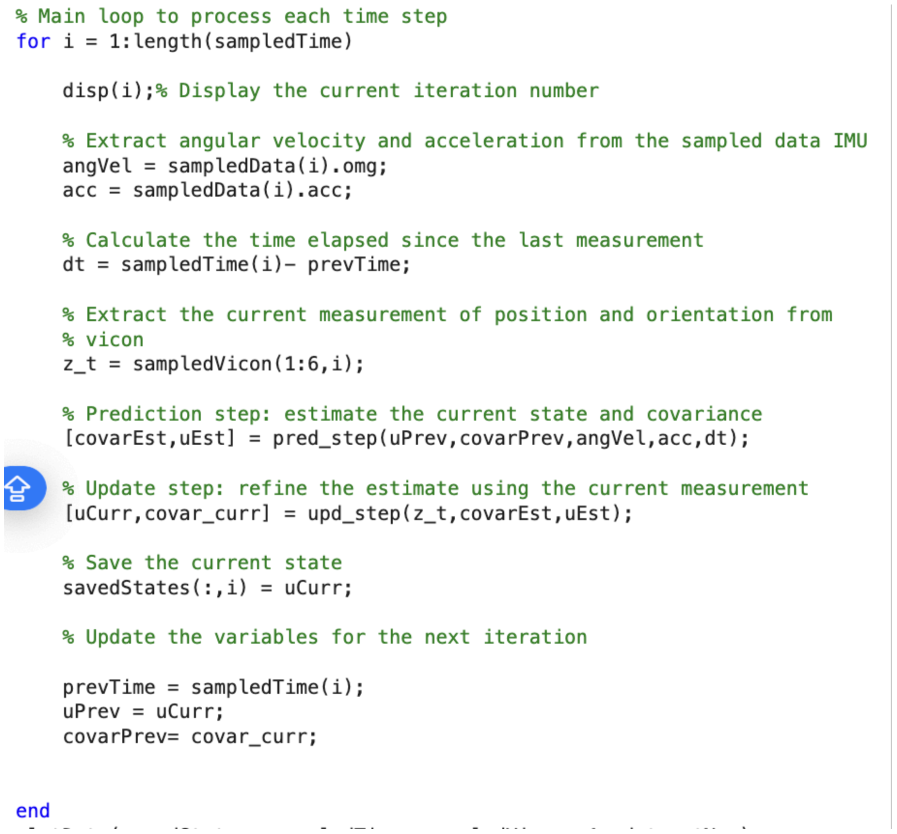


```
% Compute the Jacobian of f with respect to the state vector
J_A = jacobian(f, x);

% Compute the Jacobian of f with respect to the noise vector
J_N = jacobian(f, n);
```

The corresponding Jacobian of f wrt to state and noise vector is also computed symbolically and later the **values are just substituted** into the jacobian.



```
>> disp(J_A(10,6))
0

>> disp(J_A(6,4))
-(ng2*cos(conj(x21))*cos(conj(x22))*cos(conj(x23))^2*cos(x22)*cos(x23)^2 - wm2*cos(conj(x21)

>> disp(J_A(6,6))
0

>> disp(J_A(6,7))
0

>> |
```



```
% Main loop to process each time step
for i = 1:length(sampledTime)

    disp(i);% Display the current iteration number

    % Extract angular velocity and acceleration from the sampled data IMU
    angVel = sampledData(i).omg;
    acc = sampledData(i).acc;

    % Calculate the time elapsed since the last measurement
    dt = sampledTime(i)- prevTime;

    % Extract the current measurement of position and orientation from
    % vicon
    z_t = sampledVicon(1:6,i);

    % Prediction step: estimate the current state and covariance
    [covarEst,uEst] = pred_step(uPrev,covarPrev,angVel,acc,dt);

    % Update step: refine the estimate using the current measurement
    [uCurr,covar_curr] = upd_step(z_t,covarEst,uEst);

    % Save the current state
    savedStates(:,i) = uCurr;

    % Update the variables for the next iteration

    prevTime = sampledTime(i);
    uPrev = uCurr;
    covarPrev= covar_curr;


end
```

In the loop, for every single iteration, current state is predicted by calling the prediction script and the prediction is updated using the measurement. In part 1 the measurement is position and orientation whereas in part 2 the measurement is velocity. The current states are then saved in a matrix used for plotting later.

<u>**Prediction Model:**</u>

The prediction model is the same for part 1 and part 2.

```
% Declare global variables to access the symbolic representations
% and Jacobians of the process model and noise model

global f J_A J_N x u n ;


% Evaluate the state transition function 'f', Jacobian of 'f' with
% respect to the state 'J_A', and Jacobian of 'f' with respect to
% the noise 'J_N', substituting previous values for state, control input,
% and noise. The noise is assumed to be zero for prediction.

f_eval = double(subs(f, [x; u; n], [uPrev; [angVel; acc]; zeros(12, 1)]));
At = double(subs(J_A, [x; u; n], [uPrev; [angVel; acc]; zeros(12, 1)]));
Ut = double(subs(J_N, [x; u; n], [uPrev; [angVel; acc]; zeros(12, 1)]));

% Calculate the discrete-time state transition matrix 'Ft'
% and the input-noise-to-state matrix 'Vt' for the time step 'dt'

Ft = eye(15) + dt * At;
Vt = Ut;


% Define the process noise covariance matrix 'Q'
% and scale it by the time step 'dt' to get 'Qd'
Q = 2*eye(12);
Qd = Q * dt;

% make the mean and covariance estimate of state

uEst = uPrev + dt*f_eval;
covarEst = Ft * covarPrev* Ft'+ Vt*Qd*Vt';
```

Since f ie the state derivative, J_A the jacobian of f wrt to state, J_N jacobian of f wrt to noise were all computed symbolically in the main script as global variables. Now these are accessed and the values of **previous mean state, zero noise and control input (angular velocity and acceleration from IMU) are now substituted into the expressions.**

The noise covariance is assumed to be **2*I** and its size

The formula for the predicted mean and covariance from the previous state was the same as used in the lecture.

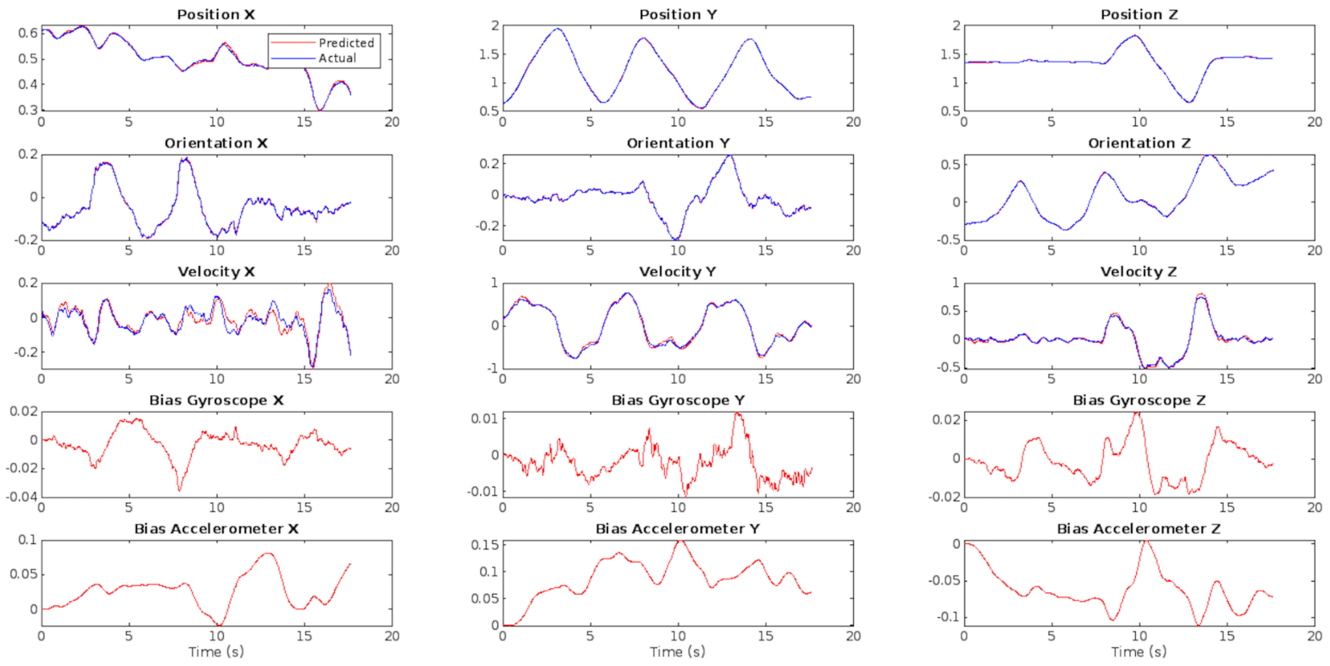
Measurement Model:

```
% Define the measurement model. Here, Ct is the measurement matrix that
% maps the state space into the measurement space. It consists of an identity
% matrix to select the relevant states, with zeros elsewhere to match dimensions.
Ct = [eye(6), zeros(6, 9)];

% Define the measurement noise covariance matrix.
Rt = 2*eye(6);

% Compute Kalman gain
A = Ct * covarEst * Ct' + Rt;
B = covarEst * Ct';
Kt = B/A;

% Update state and covariance estimate
uCurr = uEst + Kt * (z_t - Ct * uEst);
covar_curr = covarEst - Kt*Ct*covarEst;
```

As part of the measurement model in part1 and part 2, the matrix C differs since different measurements were used. The matrix C in part 1 is a composition of identity matrix (6*6) with zero matrix of 6*9 so that only the corresponding state variables are chosen. Likewise the noise covariance matrix is also different for part 1 and 2. Then the current mean and covariance of state are updated as per the formula given in the lecture after computing the kalman gain.

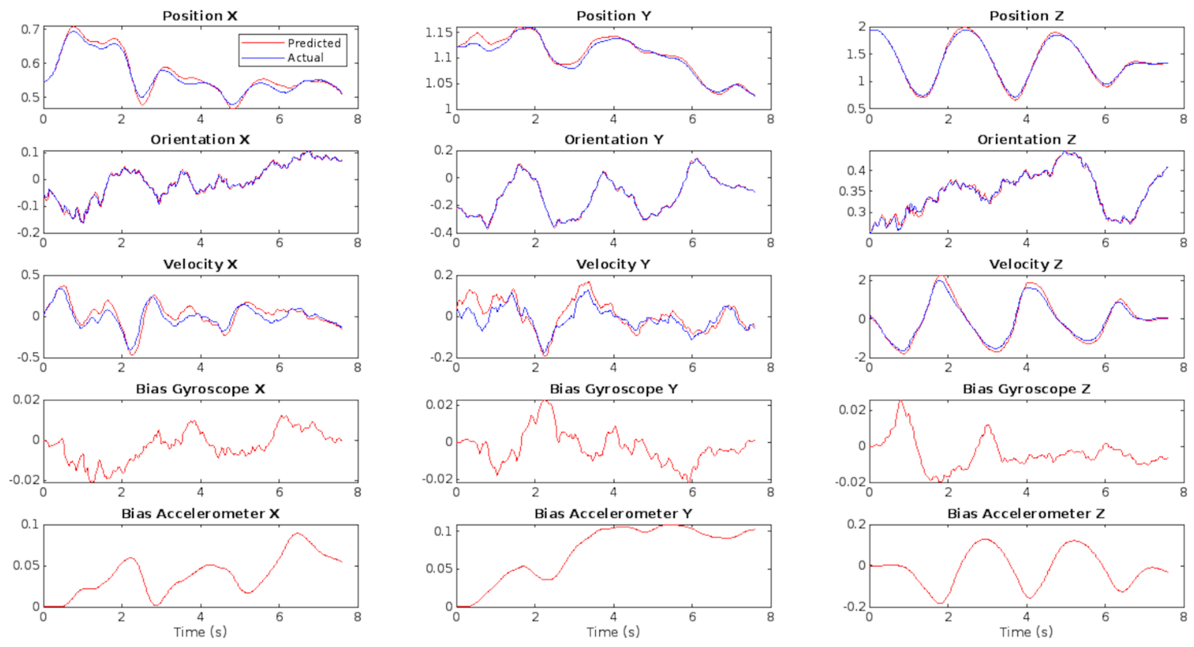
Part 1 Plots:

Dataset 1:



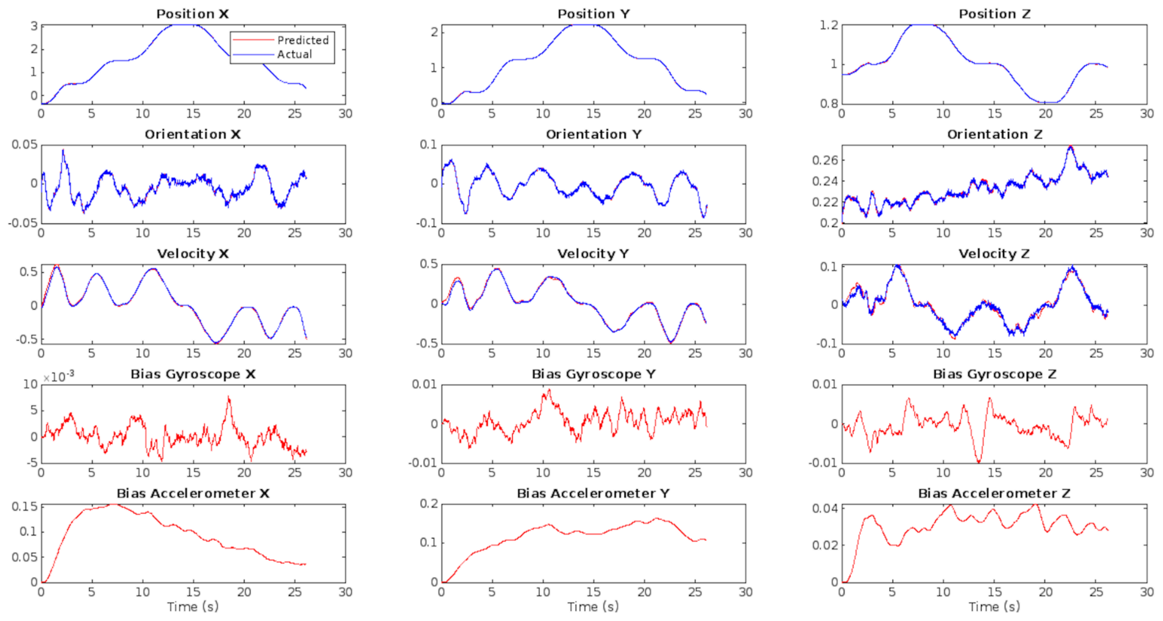
For dataset 1, the position, orientation of the state computed from the algorithm match very well with the vicon ground truth, however the velocity doesn't and there is a gap from the ground truth. This might be explained by the fact that the measurement update did not include velocity but only position and orientation. The graph shows fluctuations in bias gyroscope and bias acceleration but since this is in a very small order it can be neglected reasonably.

Dataset 4:



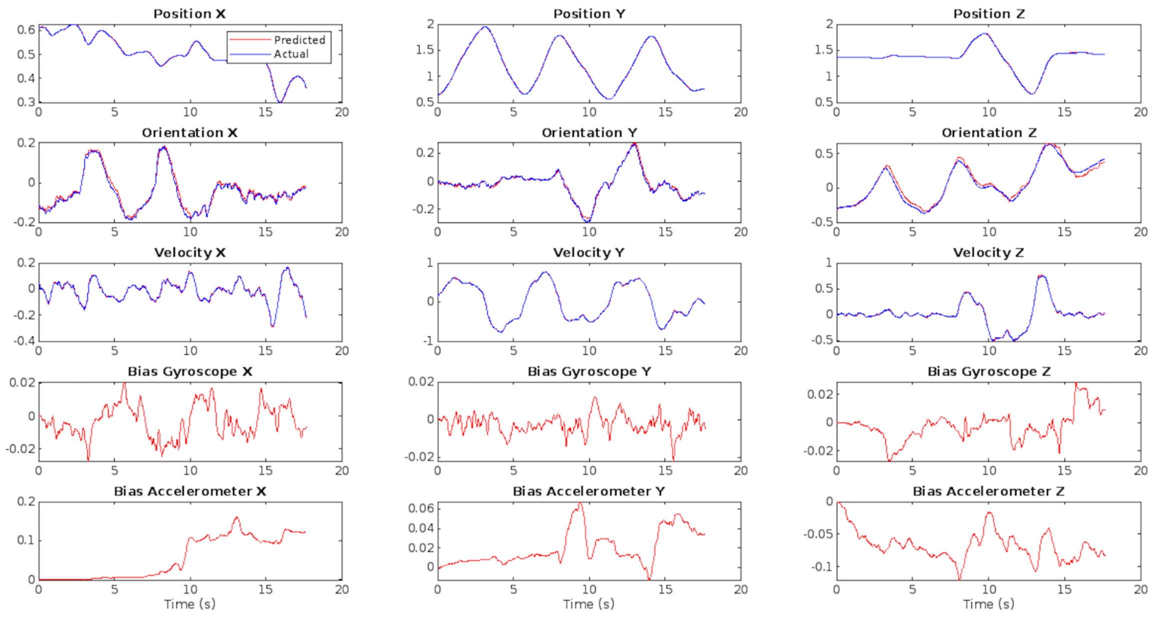
The Kalman filter estimates the position, orientation, velocity, and sensor biases reasonably well compared to the actual values but not as well as dataset 1. The position and orientation estimates closely follow the actual values with minimal deviations, indicating good tracking performance. The velocity estimates capture the overall trend but exhibit some fluctuations and noise. The bias estimates for the accelerometer in Y converge towards stable values, suggesting effective bias estimation.

Dataset 9:

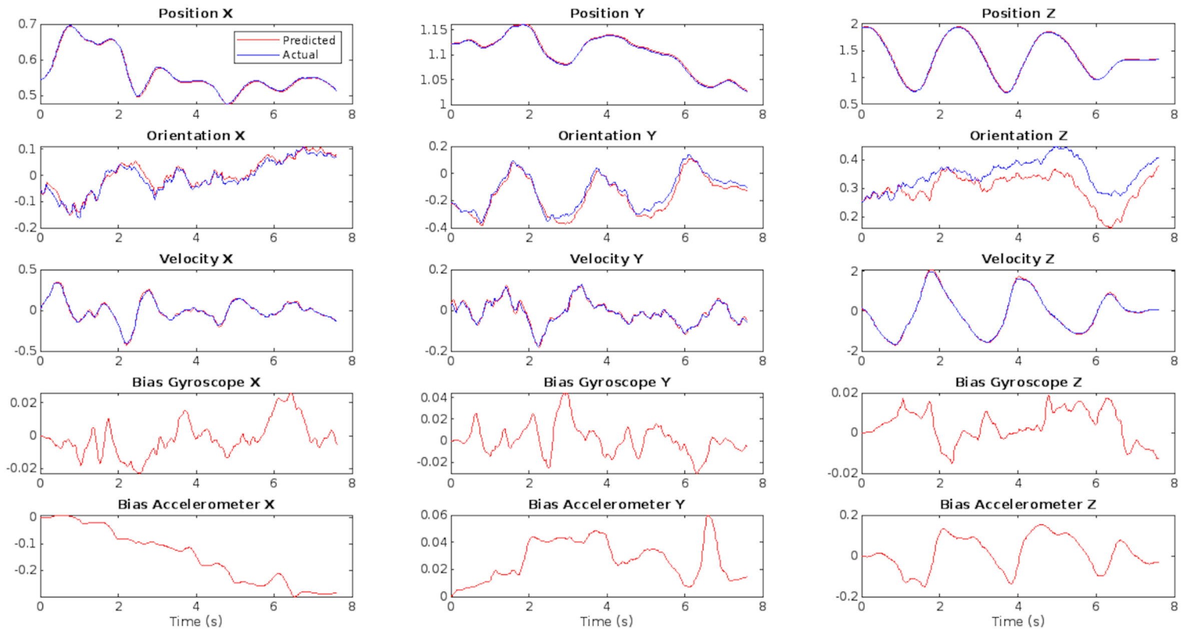


<u>**Part 2:**</u>

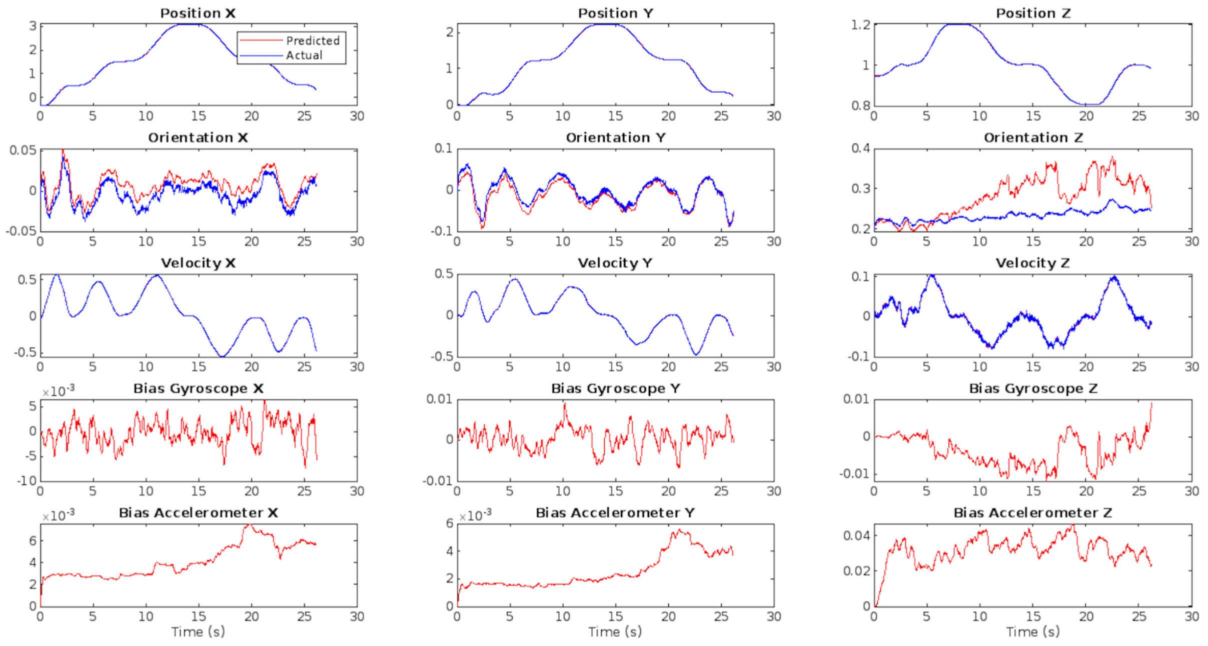
Dataset 1:



Dataset 4:


Position X
Predicted
Actual
Position Y
Position Z
Orientation X
Orientation Y
Orientation Z
Velocity X
Velocity Y
Velocity Z
Bias Gyroscope X
Bias Gyroscope Y
Bias Gyroscope Z
Bias Accelerometer X
Bias Accelerometer Y
Bias Accelerometer Z
Time (s)
Time (s)
Time (s)

Dataset 9:


Position X
Predicted
Actual
Position Y
Position Z
Orientation X
Orientation Y
Orientation Z
Velocity X
Velocity Y
Velocity Z
Bias Gyroscope X
Bias Gyroscope Y
Bias Gyroscope Z
Bias Accelerometer X
Bias Accelerometer Y
Bias Accelerometer Z
Time (s)
Time (s)
Time (s)

The predicted values match the actual values quite well for positions in all 3 datasets.
For orientations, Dataset 1 has the smoothest actual data while Dataset 9 is the noisiest, especially the Z orientation. Gyroscope biases are centered around zero but quite noisy, with Dataset 9 having the least value of range. Accelerometer biases have an offset from zero in all datasets, with Dataset 3 showing a larger, fluctuating bias especially in Z.

Overall, the Kalman filter predictions track the actual values reasonably.