1. IMU static data analysis

1.1 Accelerometer Data Analysis

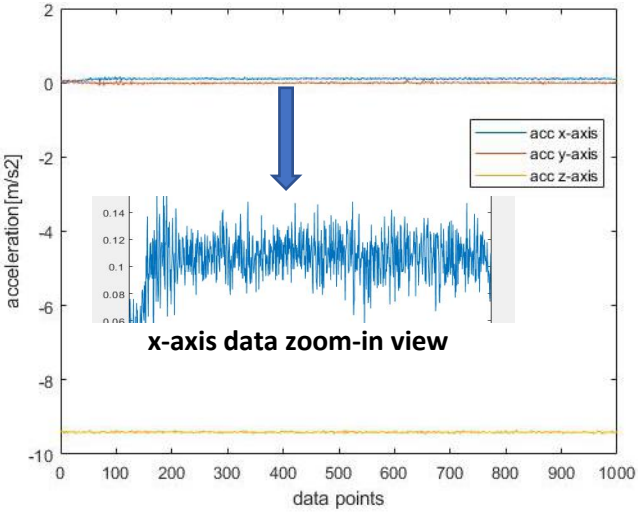


Fig 1a. Accelerometer Measurements

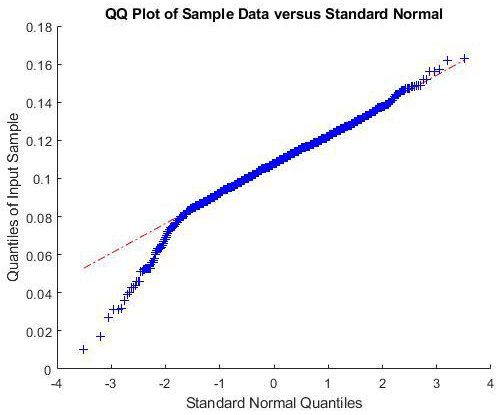


Fig 1b. QQ plot of accx data

As you can see from the plot (**Fig.1a, Fig.2a, Fig.3a**), even though the IMU is fixed on table, the measurements still contain noise.

For accelerometer(**Fig.1a**), the sensor is sensitive to mechanical noise, thus any vibration(car engine running, cooling fan operating, people walking, etc.) in the environment will affect the result. Gravity is another issue, z-axis of course has a gravity offset, but if the sensor is not perfectly level, gravity will influence x and y axis as well. The noise of accelerometer mainly depends on environment noise, for the sensor itself, it's mainly Gaussian white noise. We can use the q-q plot to see if it satisfies Gaussian distribution. From the QQ plot of x-axis acceleration data versus standard normal distribution (**Fig.1b**), we can see that the noise is mostly Gaussian.

1.2 Gyroscope Data Analysis

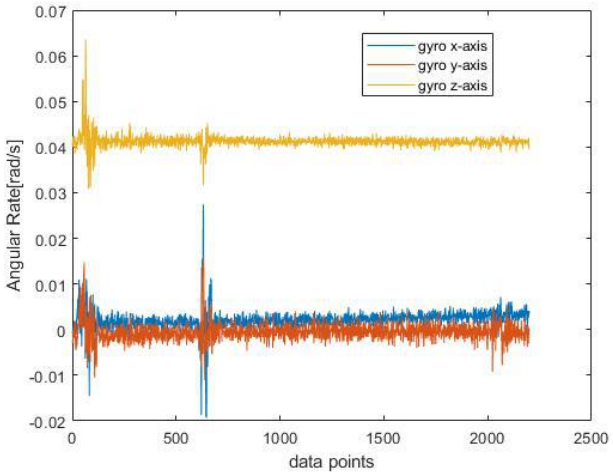


Fig 2a. Gyroscope Measurements

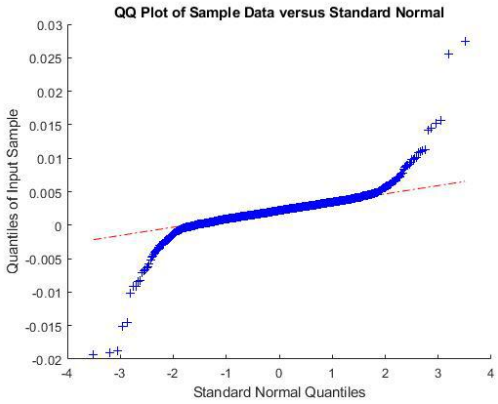


Fig 2b. QQ plot of gyrox data

For gyroscope(**Fig.2a**), the main problem is drifting and random noise. As you can see from the plot, even though gyroscope is fixed its output is not zero (bias) and will slowly drift up or down. If we integrate this data, the error will accumulate over time. Another issue is the random walk, it's nearly Gaussian white noise(**fig.2b**).

1.3 Magnetometer Data Analysis

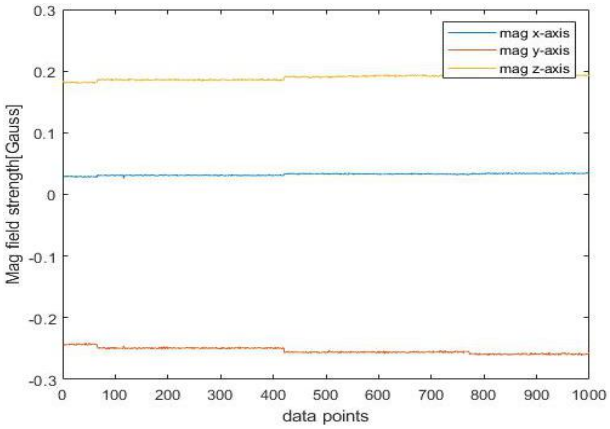


Fig 3a. Magnetometer Measurements

For magnetometer(**Fig.3a**), its reading is much cleaner and the main problem is hard and soft Iron distortions and bias. We are surrounded with metals and electronic devices, the magnetic field or merely the existence of these things can easily affect the magnetometer reading.

2. Estimating Heading

2.1 Removing hard and soft iron effects

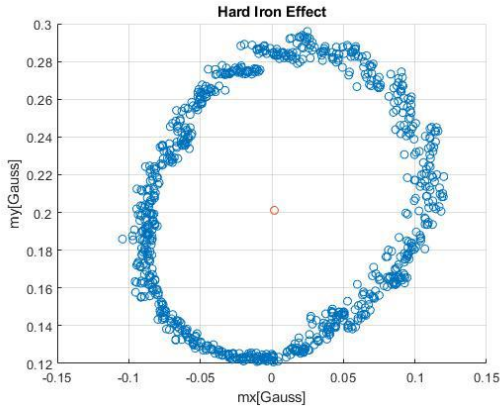


Fig 4b. Magnetometer data when going around in a circle without hard iron correction

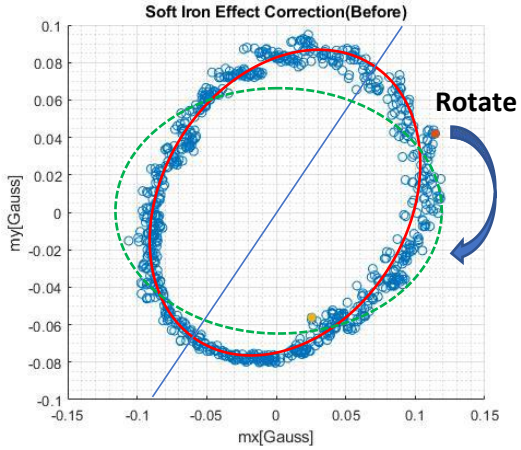


Fig 4b. Magnetometer data after hard iron correction

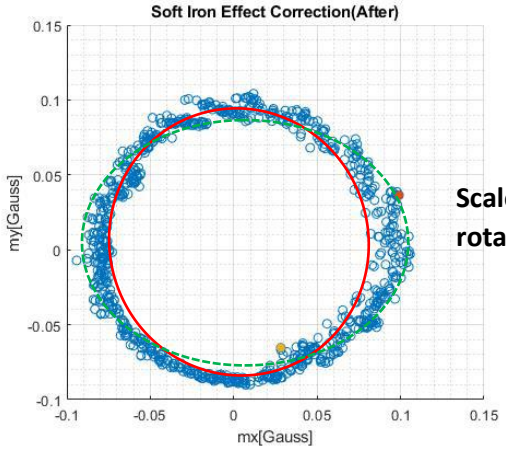


Fig 4b. Magnetometer data after soft iron correction

As mentioned earlier, the magnetometer will be affected by hard and soft iron effects. To calibrate the magnetometer, we need to plot the magnetometer data on X-Y plane. If there's no error, the data plot will be a nearly perfect circle. But in reality it's almost an ellipse with offsets. To do that the first thing we need to do is to calculate the center of the ellipse, and remove the hard iron effects. Then we need to find the axis of the ellipse and get its angle. Then rotate the whole data sets to make the axis align with x-axis. After that we apply scale factor to the ellipse and make it look like a circle. Finally, we rotate it back, and that's how we remove the soft iron effect from magnetometer reading.

2.2 Yaw angle calculation

The orientation can be calculated from magnetometer or integrated from gyroscope.

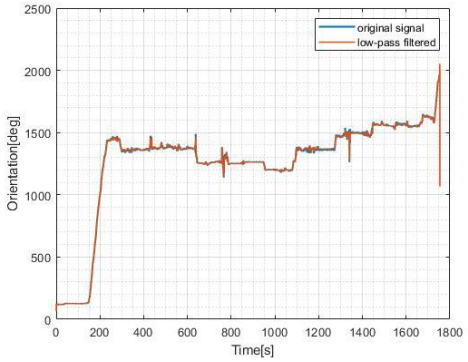


Fig 5a. yaw angle from magnetometer

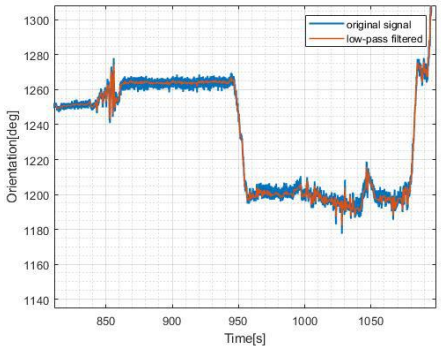


Fig 5b. yaw angle filtered

Figure 5a is the heading angle calculated from magnetometer. By convention, magnetic declination is positive when magnetic north is east of true north, and negative when it is to the west. Magnetometer data contains high-frequency noise, thus we used a low pass filter to get rid of that. In this case, because our data sampling rate is around 40Hz, a low pass filter with a passband frequency of 0.5Hz is used.

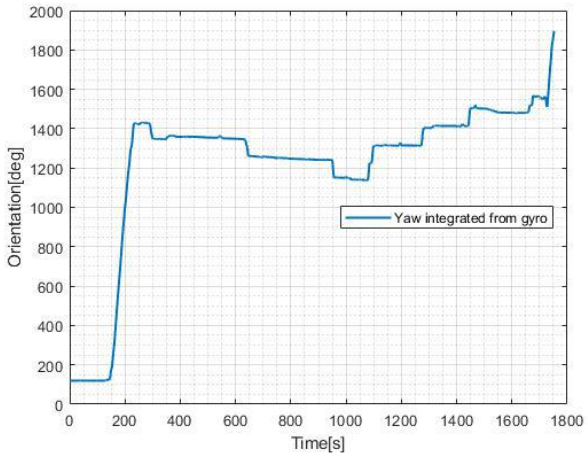


Fig 6. yaw angle from gyroscope

Figure6 is the heading angle calculated from gyroscope, the initial heading angle is set be the same as magnetometer reading to make it easier to compare them. IMU heading convention is different from magnetometer, gyroscope reading will be positive if the sensor is rotating counterclockwise around Z-axis. So when doing integration, the reading from IMU is multiplied with -1.

2.3 Complementary filter

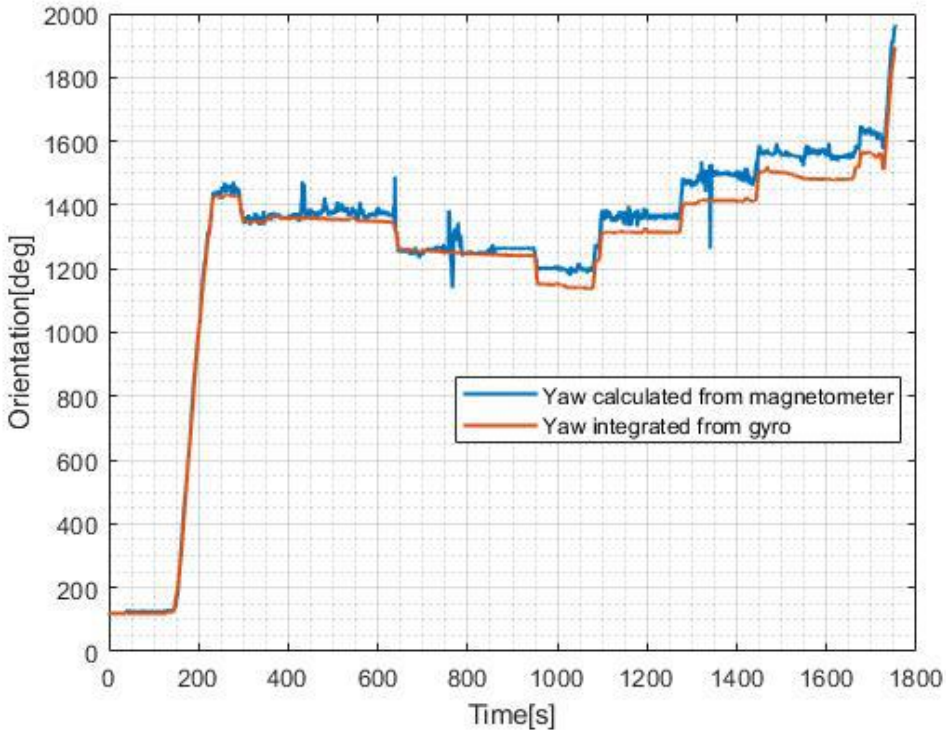


Fig 7. yaw angle from gyroscope and magnetometer

Comparing the yaw angle from magnetometer and gyroscope**(fig.7)** we can see that IMU data is much cleaner but as time goes by the integration error accumulates, causing yaw angle of IMU to slowly deviate from the yaw data of magnetometer.

In order to eliminate the high frequency noise from the magnetometer and low frequency bias from gyroscope we apply a complementary filter to merge these two signals.

The filter I used was a 1-order complementary filter:

Yaw_angle=a*yaw_integrated_from_IMU+(1-a)*yaw_from_magnetometer

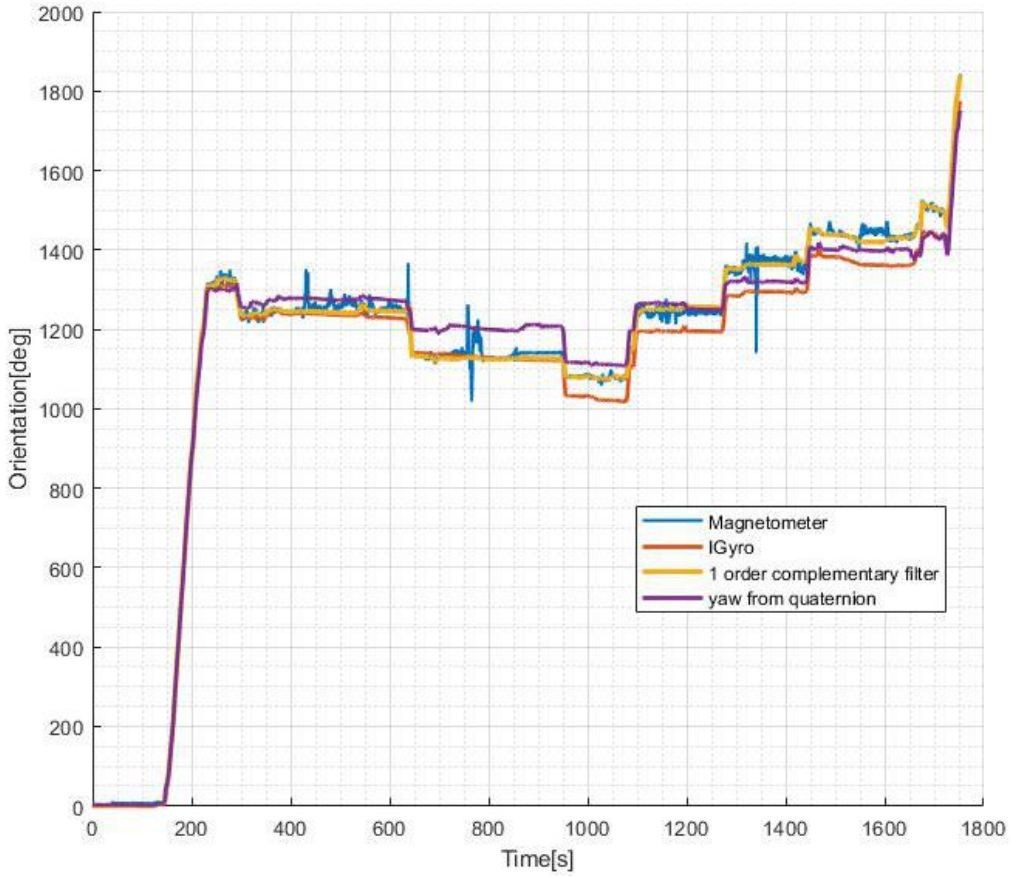


Fig 8. yaw angle with complementary filter

As you can see in **fig.8**, the **1-order complementary filtered result (yellow)** is pretty good, the accumulated error in **gyroscope data (orange)** is eliminated and there're fewer spikes in the filtered data than in the **magnetometer data (blue)**. The **complementary filtered** yaw angle mainly stays around the magnetometer data but with less noise. The sensor automatically integrate the yaw angle inside, purple line is from sensor direct quaternion output. Yet the sensor is not calibrated at the beginning so it will be affected by multiple things such as hard iron, soft iron effects, gyro bias, accelerometer offsets, etc.

2.3 Further polishing of data

To make the yaw angle more accurate, we can take advantage of some common knowledge. We assume that the roads we drove on are either parallel or perpendicular to each other, which means the **yaw angle can only change pi or -pi at each turn**. I applied a turning detector to the yaw data, it basically takes average of some data points around current time and compare it with the average yaw angle collected a few seconds ago, if the difference is around pi, we round it to pi, vice versa. As you can see from figure, the **adjusted complementary filtered yaw angle (black)** is more reasonable and will make the estimation error to be smaller.

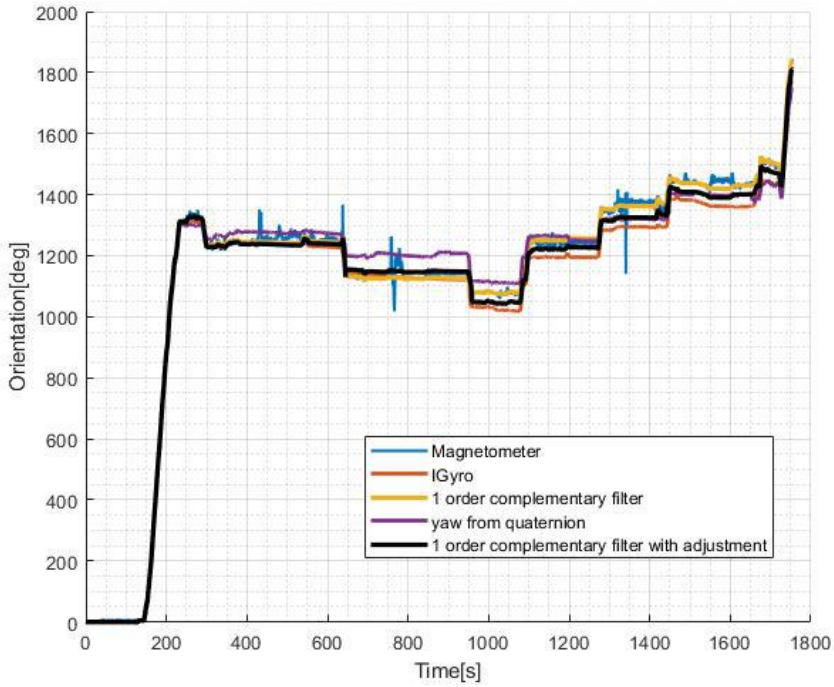


Fig 8. yaw angle with adjustment

3. Estimating Forward Velocity

3.1 Accelerometer Calibration

. As mentioned before, accelerometer can be affected by its posture and bias. So we used a model as following to describe the tranformation, in which **At** is the actual acceleration and **Am** indicates measurement:

$$A_t = \begin{bmatrix} A_x \\ A_y \\ A_Z \end{bmatrix} = R_{[3*3]} \begin{bmatrix} Scale_x & 0 & 0 \\ 0 & Scale_y & 0 \\ 0 & 0 & Scale_z \end{bmatrix} \begin{bmatrix} A_{mx} - offset_x \\ A_{my} - offset_y \\ A_{mz} - offset_z \end{bmatrix}$$

The tranformation can be further simplified as a single 3 by 4 matrix

$$\begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} = \begin{bmatrix} C_1 & C_2 & C_3 & C_{off_x} \\ C_4 & C_5 & C_6 & C_{off_y} \\ C_7 & C_8 & C_9 & C_{off_z} \end{bmatrix} \begin{bmatrix} A_{mx} \\ A_{my} \\ A_{mz} \\ 1 \end{bmatrix}$$

To solve the transformation matrix we take a transpose on both sides:

$$\begin{bmatrix} A_{mx} & A_{mx} & A_{mx} & 1 \end{bmatrix} \begin{bmatrix} C_1 & C_4 & C_7 \\ C_2 & C_5 & C_8 \\ C_3 & C_6 & C_9 \\ C_{off_x} & C_{off_y} & C_{off_z} \end{bmatrix} = \begin{bmatrix} A_x & A_y & A_z \end{bmatrix}$$

This equation can be solved by least squares method.

To solve that we need to place the IMU facedown on its 6 directions and take 1000 data points on each direction. So we will get a 6000 by 4 measurement matrix **Am** and a 6000 by 3 actual value matrix **At.** The sequence will be xdown, xup, ydown,yup,zup and z down.

$$\boldsymbol{A_t} = \begin{bmatrix} ones(1000,1) * 9.8 & 0 & 0 \\ ones(1000,1) * -9.8 & 0 & 0 \\ 0 & ones(1000,1) * 9.8 & 0 \\ 0 & ones(1000,1) * -9.8 & 0 \\ 0 & 0 & ones(1000,1) * 9.8 \\ 0 & 0 & ones(1000,1) * -9.8 \end{bmatrix}$$

Once we got the measurement, we can simply calculate transformation matrix R:

$$R = (\boldsymbol{A_t}^T \boldsymbol{A_t})^{-1} \boldsymbol{A_t}^T \boldsymbol{A_m} = \begin{bmatrix} 0.9987 & 0.0364 & -0.0017 \\ -0.0255 & 1.0017 & 0.0716 \\ -0.0008 & -0.0023 & 1.00180 \\ -0.0972 & -0.1051 & -0.4489 \end{bmatrix}$$

The result is shown in **fig.9:**

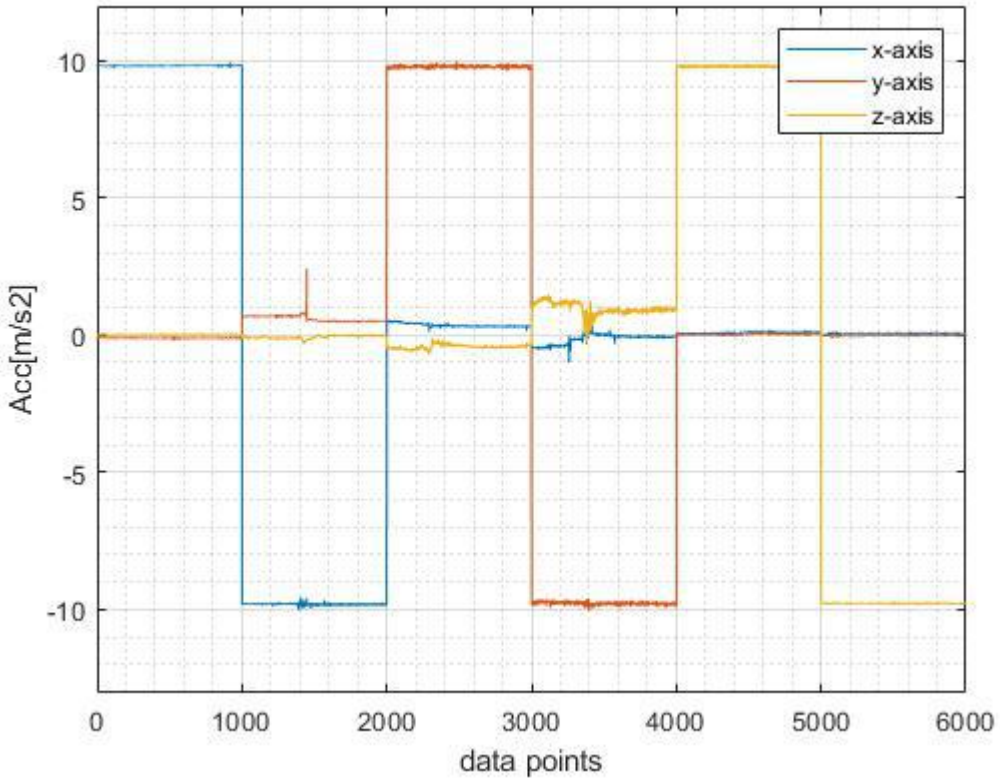


Fig 9a. IMU data not calibrated

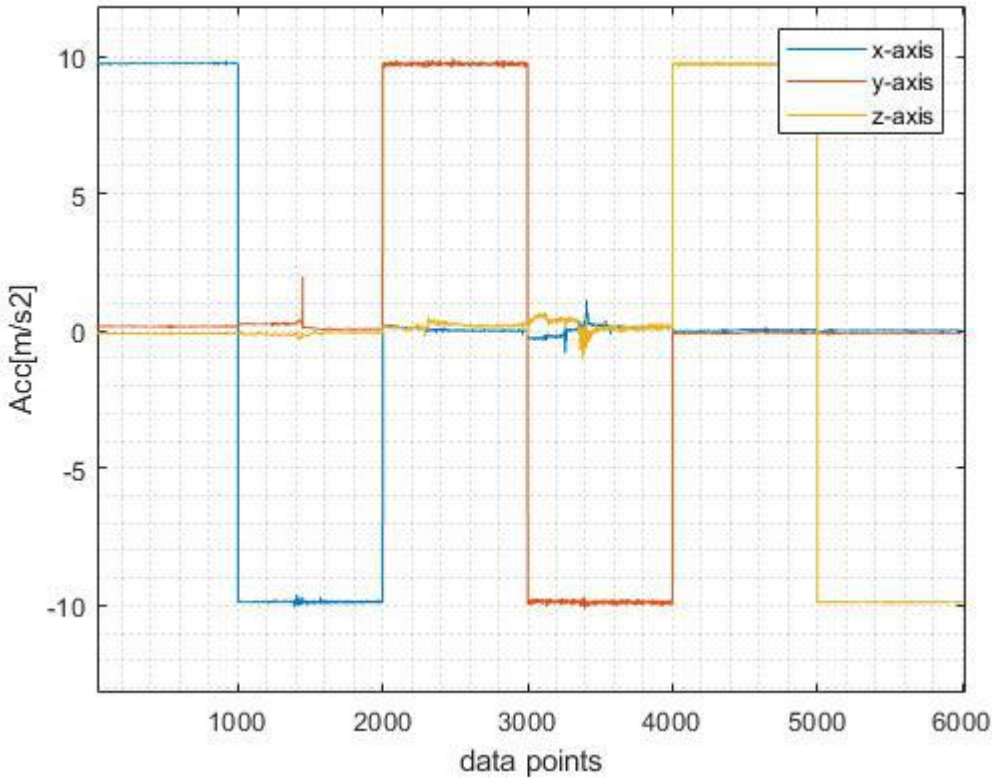


Fig 9a. IMU data calibrated

3.2 Acceleration data integration

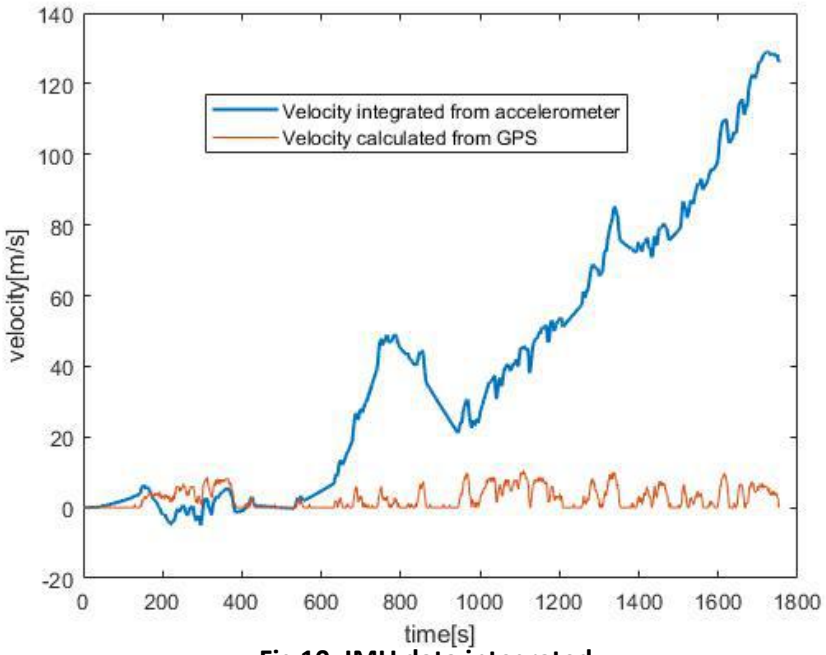


Fig 10. IMU data integrated

Direct Integration of the acceleration data is inaccurate, the error will accumulate and eventually the speed will blow up. One way is to subtract the mean value of acceleration from the data, but our data is collected when there was a lot of traffic, so there's a lot of sudden acceleration and deceleration which caused large error in our data (**fig10**).

To eliminate that error, I made **a few assumptions:**

1. The error introduced is due to sudden movement of the vehicle, the process of acceleration and deceleration can cause slight change of sensor pitch angle and thus we will have additional gravity portion inside our acceleration data.

2. Velocity should always be positive because we are always going forward.

Based on these assumptions we can start filtering the data.

First thing I did was designing a detector for vehicle movement. It's commonsense that a car vibrates more when it's moving, especially on roads of Boston.

Thus I took a high-pass filter on the Z-axis data to get the high frequency noise, then I calculate the variance of that noise inside a window of 200 data points (**fig.11** bottom). If this value exceeds a threshold(red line), it means the vehicle is vibrating a lot, indicating the movement.

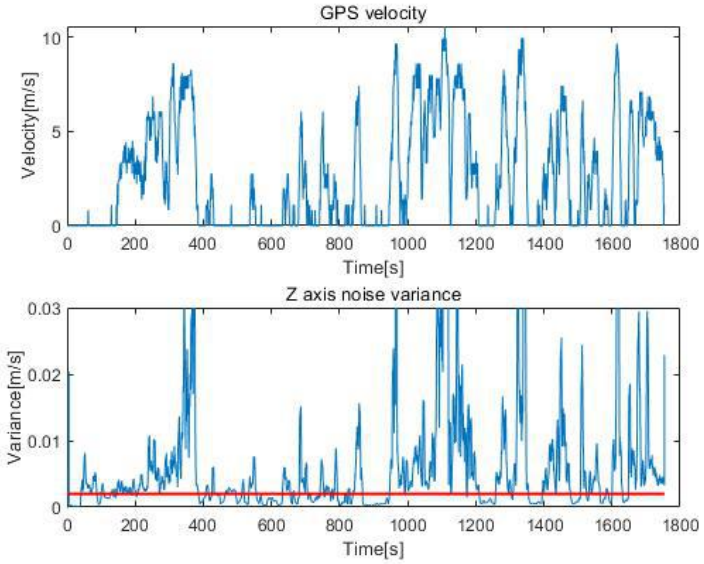


Fig 11. Movement detector

As you can see in figure, this detector can perfectly detect the movements of the vehicle. **Red** points means the vehicle is stopped, **blue** means the vehicle is moving.

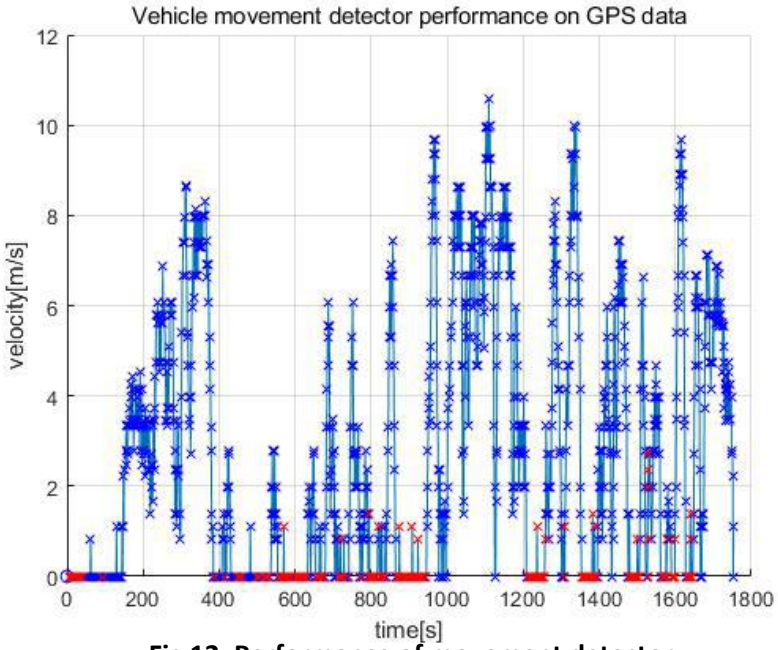


Fig 12. Performance of movement detector

With the help of the this detector we can know when to set velocity to be zero, in order to regulate the integration error of velocity from accelerometer.

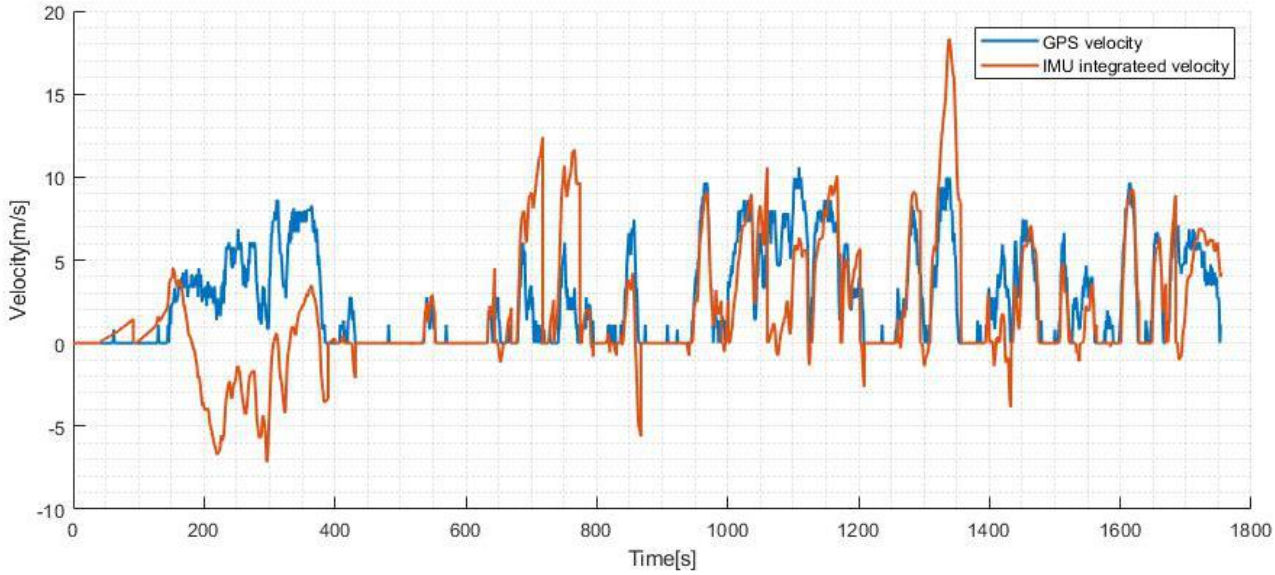


Fig 13. Velocity regulated with movement detector

After error regulation, we can see that the velocity is almost accurate, but there exists some parts with negative velocity due to sudden deceleration error. This means the deceleration measurement is much larger than actual value. To regulate that I simply applied a scale factor to the previous data to shift it up above zero:

When detected negative velocity, applied scale fator α to its previous 200 data points, in which:

$$\alpha = \frac{\text{mean}(\text{data}(\text{i} - 200{:}\text{i})) - \text{velocity}}{\text{mean}(\text{data}(\text{i} - 200{:}\text{i}))}$$

After the adjustment, the velocity is very close to the GPS value.

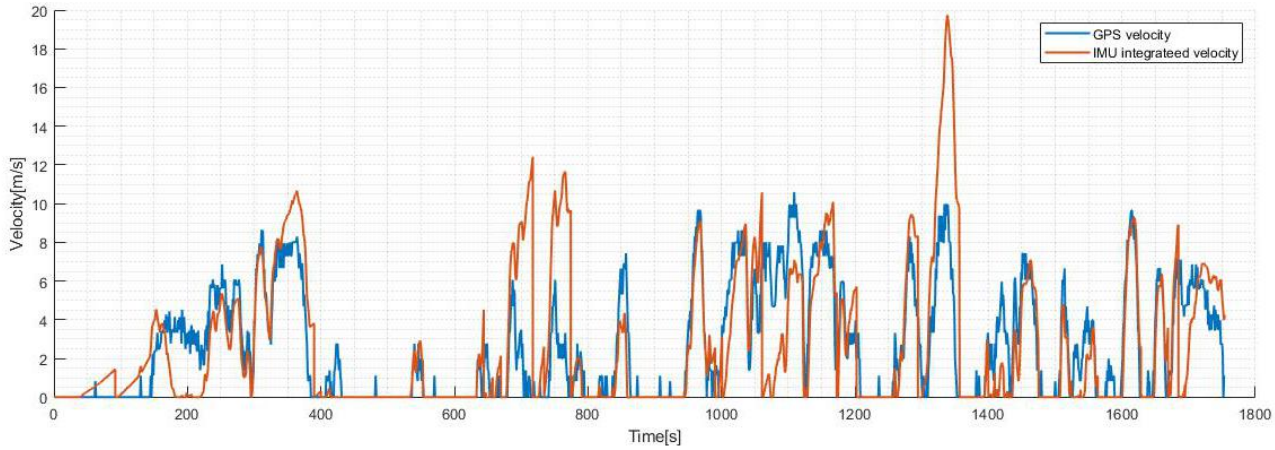


Fig 14. Velocity regulated with adjustment

3. 3 Dead-reckoning with IMU

3.3.1 Comparing $\dot{y}_{observe}$ with $\omega\dot{x}$

$$\ddot{y}_{observe} = \ddot{Y} + \omega\dot{X} + \dot{\omega}x_c$$

In which $\ddot{Y}$ and x_c are assumed to be zero thus we have $\ddot{y}_{observe} = \omega\dot{X}$.

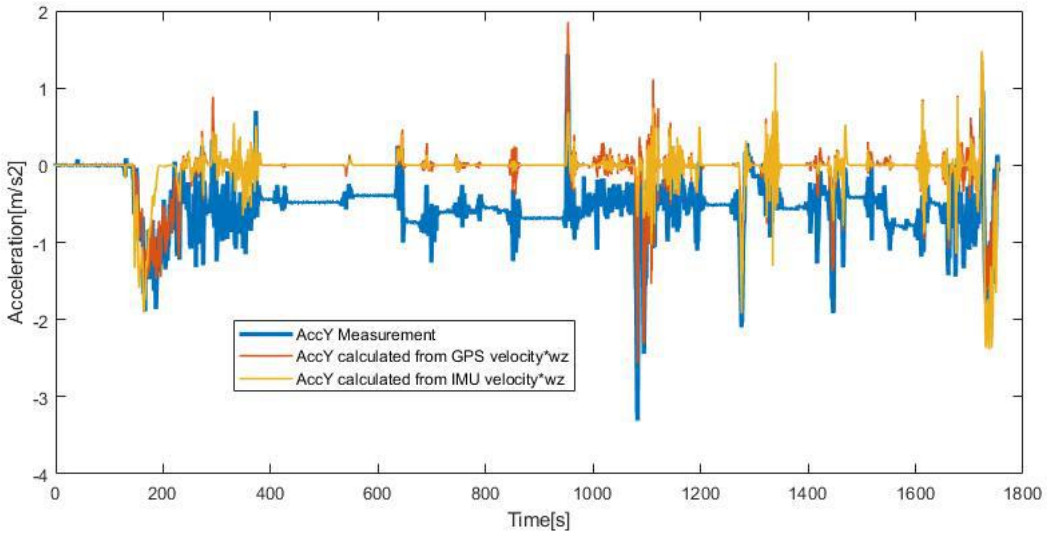


Fig 14. Comparison of $\dot{y}_{observe}$ and $\omega\dot{x}$

As is shown in figure, $\ddot{y}_{observe}$ and $\omega\dot{X}$(GPS) generally matches well, $\omega\dot{X}$(IMU) has larger error which is expected since the measurement is not very accurate. We notice that there's an offset between $\ddot{y}_{observe}$ measurement and calculated y-axis acceleration, which is caused by ignoring $\dot{\omega}x_c$, the distance between vehicle inertia center and IMU. And the IMU y-axis positive is opposite from normal convention, thus we multiplied our calculation by -1.

3.3.2 Trajectory Estimation

$$\ddot{x}_{absolute} = {}^{V}_{W}R^{-1}\ddot{X} = {}^{V}_{W}R^{-1}\left(\ddot{x}_{observe} + \omega\dot{Y} + \omega^2 x_c\right),$$

$$in\ which\ \ \dot{Y} = 0, x_c = 0$$

$\ddot{x}_{abs}$ is x-axis acceleration under world coordinate frame, and $\ddot{X}$ is measurement under vehicle coordinate frame. So taking an integration and applying zero velocity initial condition we have:

$$\begin{bmatrix} v_{xabs} \\ v_{yabs} \end{bmatrix} = \dot{x}_{absolute} = {}^{V}_{W}R^{-1}\dot{X}$$

$$in\ which\ {}^{V}_{W}R = \mathrm{eul2rotm}([0,0,-\mathrm{yaw} + \theta_{\mathrm{offset}}])$$

${}^{V}_{W}R$ is the rotation matrix from world frame to vehicle frame, in this case, $\theta_{\mathrm{offset}} = 130°(deg)$ align the IMU data with GPS measurement under utm coordinate.

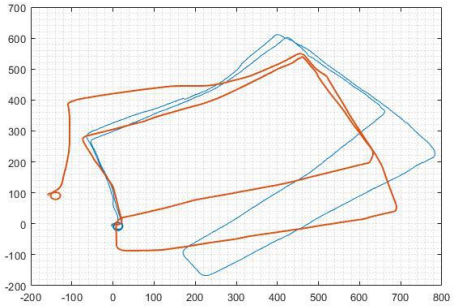


Fig 15a. dead-reckoning with GPS velocity

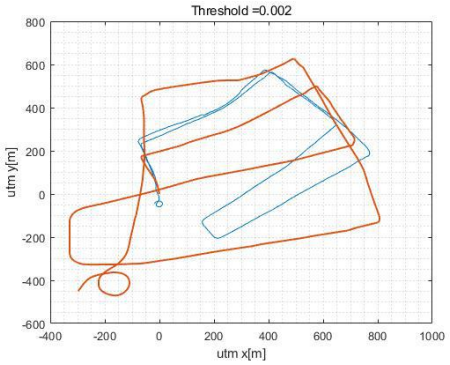
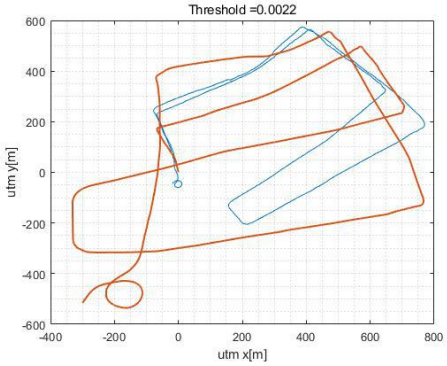


Fig 15b



Fig

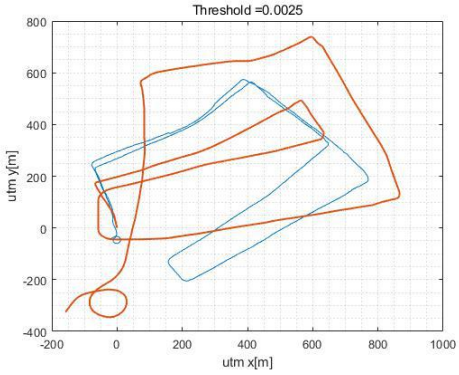


Fig 15d

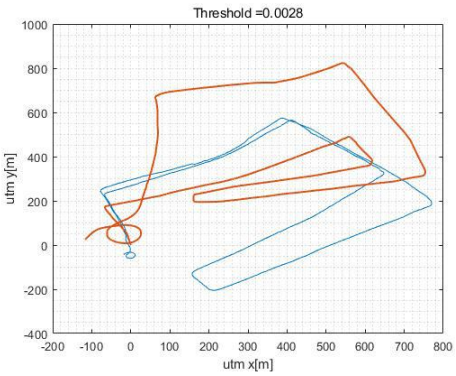


Fig 15e

Fig 15b-e. dead-reckoning with IMU velocity under different thresholds

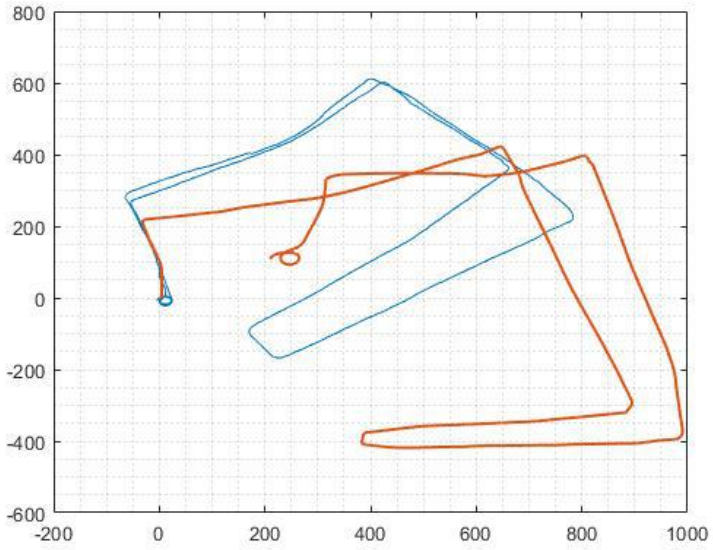
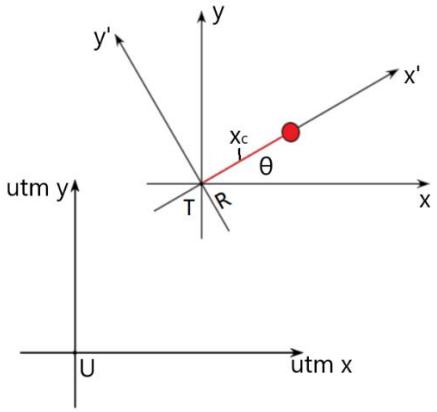


Fig 15f Trajectory with IMU velocity with variable thresholds: 0.0012(t<400s),0.0028(400s-1600s),0.0032(t>1600s)

From **fig15b-f** we can see that the yaw angle integration error together with velocity integration error eventually pushes the endpoint of our route away form the actual position. Our yaw estimation is close to accurate value, moving at velocity measured by GPS, the trajectory is very close to the actual trajectory. The error is mainly introduced by inaccurate velocity measurement, comparing b,c with a, we can see that velocity error is the dominant factor in accurate trajectory estimating. By tweaking the threshold value, we can make slight adjustment on the trajectory.

With GPS accurate velocity we can achieve an error around 120m, with IMU velocity with adjustments, the best is around 100-400m.

3.3 **Xc estimation**



$$v^U_{sensor} = v^U_{car} + \omega \times \rho^R_{sensor}$$

$$a^U_{sensor} = a^U_{car} + \dot{\omega} \times \rho^R_{sensor} + \omega \times (\omega \times \rho^R_{sensor})$$
$$^R_T R^T a^R_{sensor} = a^U_{car} + \dot{\omega} \times \rho^R_{sensor} + \omega \times (\omega \times \rho^R_{sensor})$$

$$ {}^{R}_{T}R^{T}\begin{bmatrix} a_{imux} \\ a_{imuy} \\ 0 \end{bmatrix} = \begin{bmatrix} a_{utmx} \\ a_{utmy} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\omega} \end{bmatrix} \times \begin{bmatrix} x_c \\ 0 \\ 0 \end{bmatrix}^{R} + \begin{bmatrix} 0 \\ 0 \\ \omega \end{bmatrix} \times \left(\begin{bmatrix} 0 \\ 0 \\ \omega \end{bmatrix} \times \begin{bmatrix} x_c \\ 0 \\ 0 \end{bmatrix}^{R} \right) $$

$$ \begin{bmatrix} a_{imux} \\ a_{imuy} \\ 0 \end{bmatrix}^{R} = {}^{R}_{U}R \begin{bmatrix} a_{utmx} \\ a_{utmy} \\ 0 \end{bmatrix}^{U} + \begin{bmatrix} 0 \\ \dot{\omega}x_c \\ 0 \end{bmatrix}^{R} + \begin{bmatrix} -\omega^2 x_c \\ 0 \\ 0 \end{bmatrix}^{R} $$

$$ \begin{bmatrix} a_{imux} \\ a_{imuy} \\ 0 \end{bmatrix}^{R} - {}^{R}_{U}R \begin{bmatrix} a_{utmx} \\ a_{utmy} \\ 0 \end{bmatrix}^{U} = \begin{bmatrix} -\omega^2 \\ \dot{\omega} \\ 0 \end{bmatrix}^{R} x_c $$

$$ B = Ax_c \text{ , this gives } x_c = (\mathrm{A}^{T}\mathrm{A})^{-1}\mathrm{A}^{T}B $$

in which $A = \begin{bmatrix} a_{imux} \\ a_{imuy} \\ 0 \end{bmatrix}^{R} - {}^{R}_{U}R \begin{bmatrix} a_{utmx} \\ a_{utmy} \\ 0 \end{bmatrix}^{U}$, $B = \begin{bmatrix} -\omega^2 \\ \dot{\omega} \\ 0 \end{bmatrix}$

$$ {}^{R}_{U}R = \mathrm{eul2rotm}([0,0,-\mathrm{yaw}]) $$

$$ x_c = 0.0084m; $$

This value is small but reasonable because we fixed our sensor on the dash board, it's very close to the center of mass of the vehicle.