LAB 4

Navigation with IMU and Magnetometer (1 Late day)

Shakeel Ahamed | NUID 002199480 | Team 14 | Section 1

Objective:

The aim of this lab is to build a navigation stack by using both GPS and IMU sensors, we'll also be using sensor fusion to club data from different sensors to get some relevant parameters.

IMU Sensor:

Inertial Motion Unit Sensor is capable of measuring orientation data and to achieve this it uses a combination of sensors.

There are 2 variants of IMU Sensors:

- 6 DOF
- 9 DOF

The device we'll be working with is a Vectornav VN100 IMU which is a 9 Degrees of Freedom device that has 3 sensors:

- Accelerometer:
  Accelerometers measure linear acceleration along all 3 axes.
- Gyroscope:
  Gyroscopes measure angular velocity along all 3 axes.
- Magnetometer:
  Magnetometers measure magnetic fields along all 3 axes.
- Sensor Fusion| Roll, Pitch, Yaw:
  We obtain roll, pitch and yaw from these 3 sensor readings, Accelerometer gives the roll and pitch but not reliable yaw, the sensor uses Magnetometer readings to calculate yaw.

GPS Sensor:

Global Positioning System Sensor is capable of measuring Latitude (with direction), Longitude (with direction) and Altitude and we'll be using this data to calculate the UTM Northing, UTM Easting values that are relevant to this Lab.

Method:

The steps taken to get our readings:

- Configured Ubuntu:
  Changed some parameters as guided in the previous Lab word document to make the VN-100 sensor work as intended with our OS.
- Configured Device Baud rate:
  set Baud rate to 115200 via minicom
- Device driver:
  A python device driver was written to parse data from the sensor and publish it as a sensor.msg, outputting 3x Headers, 3x Gyro, 3x Accelerometer, 3x Magnetometer, Yaw, Pitch, Roll and 4x Quaternion readings.
  Another python device driver was used from lab 1 to get the GPS reading, Latitude, Longitude, Altitude, UTM easting, UTM northing are the relevant outputs from it.
- Setting up the sensors on the nuance university vehicle:
  We kept the GPS sensor on the roof so that it does not get signal interference as opposed to keeping it inside the car.
  The IMU sensor was taped approximately at the center of mass of the car near the gear shifter above the storage room.
- Data Collection / Rosbag:
  Rosbag was initiated to take both GPS and IMU topics.
  Car pathing is mentioned further in the report.

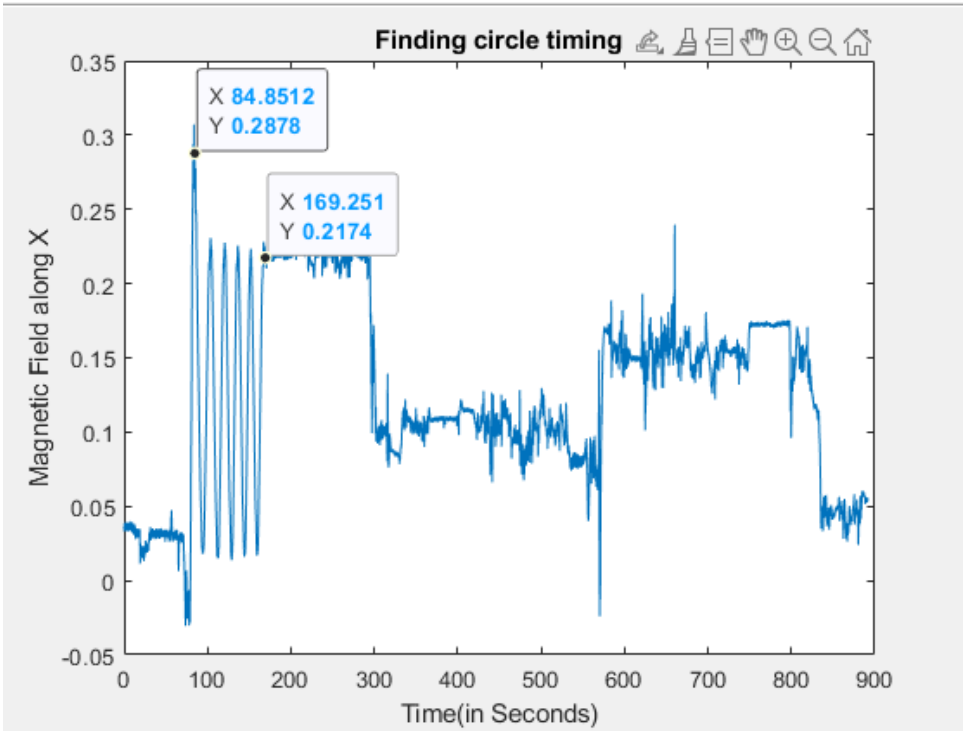
Data:

- Data Pathing:
  The pathing of the car were as follows:
  -> Standstill for over a minute near Ruggles roundabout.
  -> 4 rounds around the roundabout.
  -> Exit roundabout and drive to Huntington Ave.
  -> Go left on Huntington Ave and drive for about 4 minutes
  -> Took a U-turn
  -> Retraced back all the way to Ruggles station on the opposite side of the road.
- Convert the ROSBag into CSV readable files:
  The ROSBag data received was turned into a CSV file using a python script (saved as bagreader.py)
- Parsing out the parameters needed:
  The Gyroscope, Accelerometer, Magnetometer, Latitude, Longitude, UTM Easting, UTM Northing readings were parsed out and used for graphing and analysis as follows.

Analysis:

Part1: Estimate the heading (Yaw)

We try to eyeball and get accurate timing to parse out the roundabout data as follows:



Based on this we assume the roundabout entry exit to be around 85s -> 170s and parse the data out for that region to proceed with the “hard-iron” and “soft-iron” corrections.

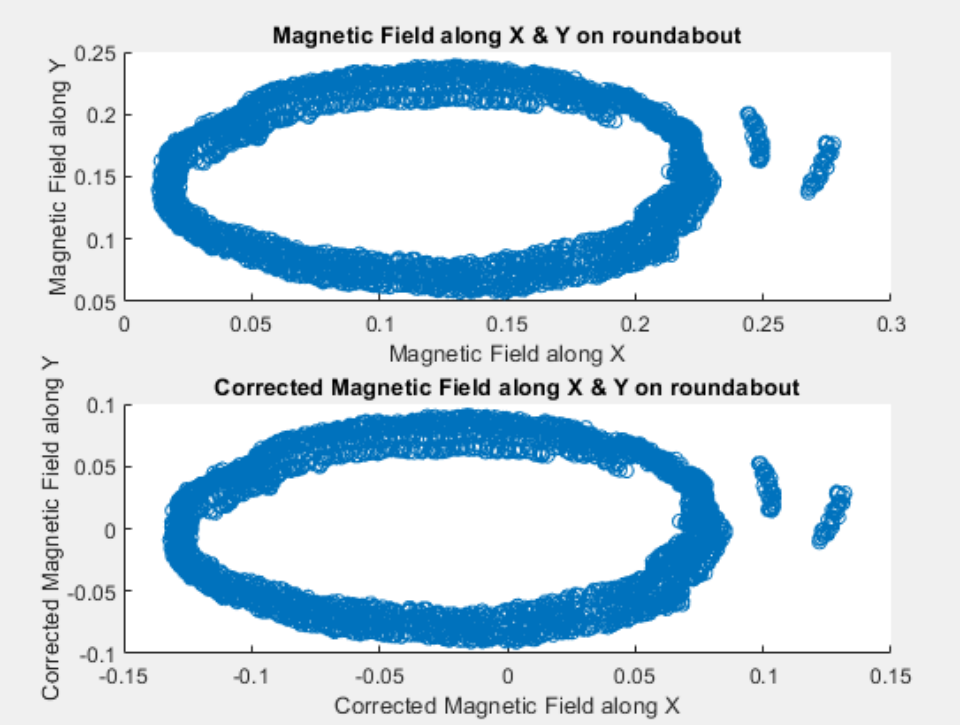
Hard-iron effect:

Interference can be caused by ferromagnetic material or equipment in the magnetometer's vicinity. If the magnetic field is permanent it is called “hard iron”.

We use the following pseudocode to fix Hard iron:

```
offset_x = (max(x) + min(x)) / 2
offset_y = (max(y) + min(y)) / 2
offset_z = (max(z) + min(z)) / 2

corrected_x = sensor_x - offset_x
corrected_y = sensor_y - offset_y
corrected_z = sensor_z - offset_z
```

Roundabout data before and after fixing hard-iron, we only consider Magnetic field along X and Y and ignore Z.



The scaling varies due to the corrections.

Soft-iron effect:

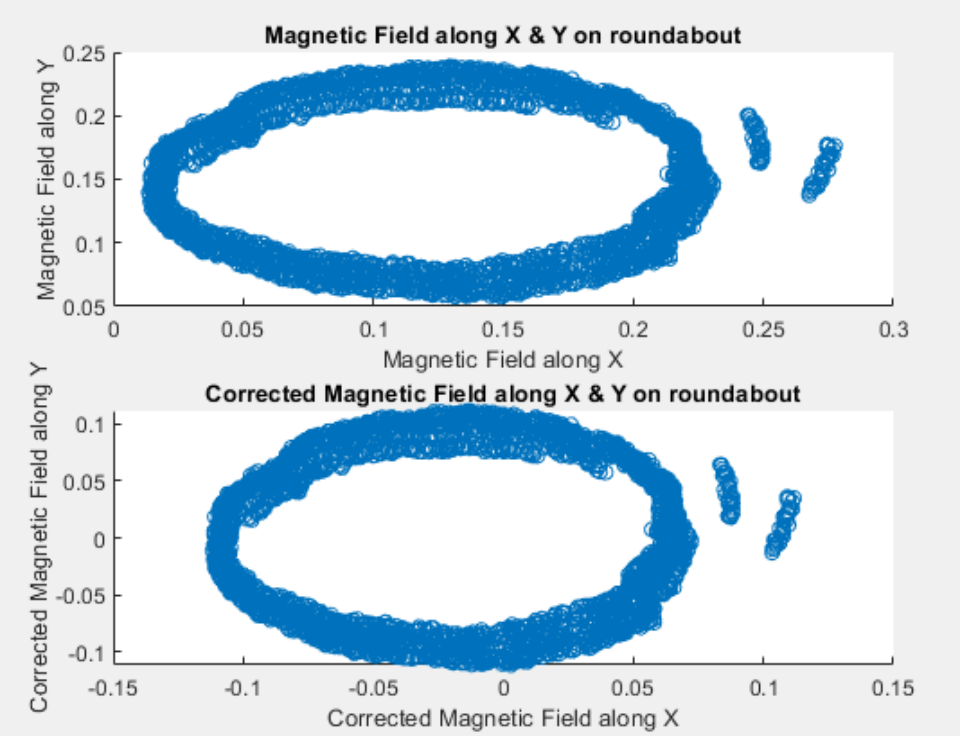
Soft iron distortion is the result of material that distorts a magnetic field but does not necessarily generate a magnetic field itself. For example iron (the metal) will generate a distortion but this distortion is dependent upon the orientation of the material relative to the magnetometer.

Pseudocode to fix soft-iron in addition to hard-iron:

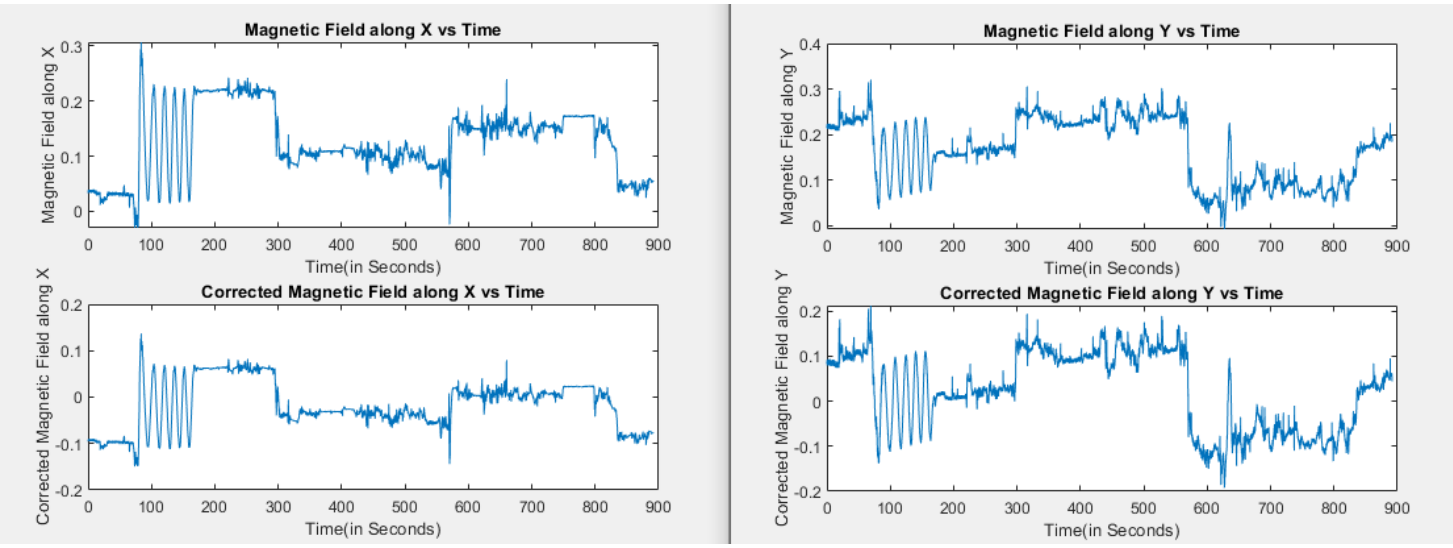
```
avg_delta_x = (max(x) - min(x)) / 2
avg_delta_y = (max(y) - min(y)) / 2
avg_delta_z = (max(z) - min(z)) / 2

avg_delta = (avg_delta_x + avg_delta_y + avg_delta_z) / 3

scale_x = avg_delta / avg_delta_x
scale_y = avg_delta / avg_delta_y
scale_z = avg_delta / avg_delta_z

corrected_x = (sensor_x - offset_x) * scale_x
corrected_y = (sensor_y - offset_y) * scale_y
corrected_z = (sensor_z - offset_z) * scale_z
```

Hard-iron and Soft-iron effects removed:

Corrected Roundabout Data:



The following are the plots for full magnetometer data before and after corrections.



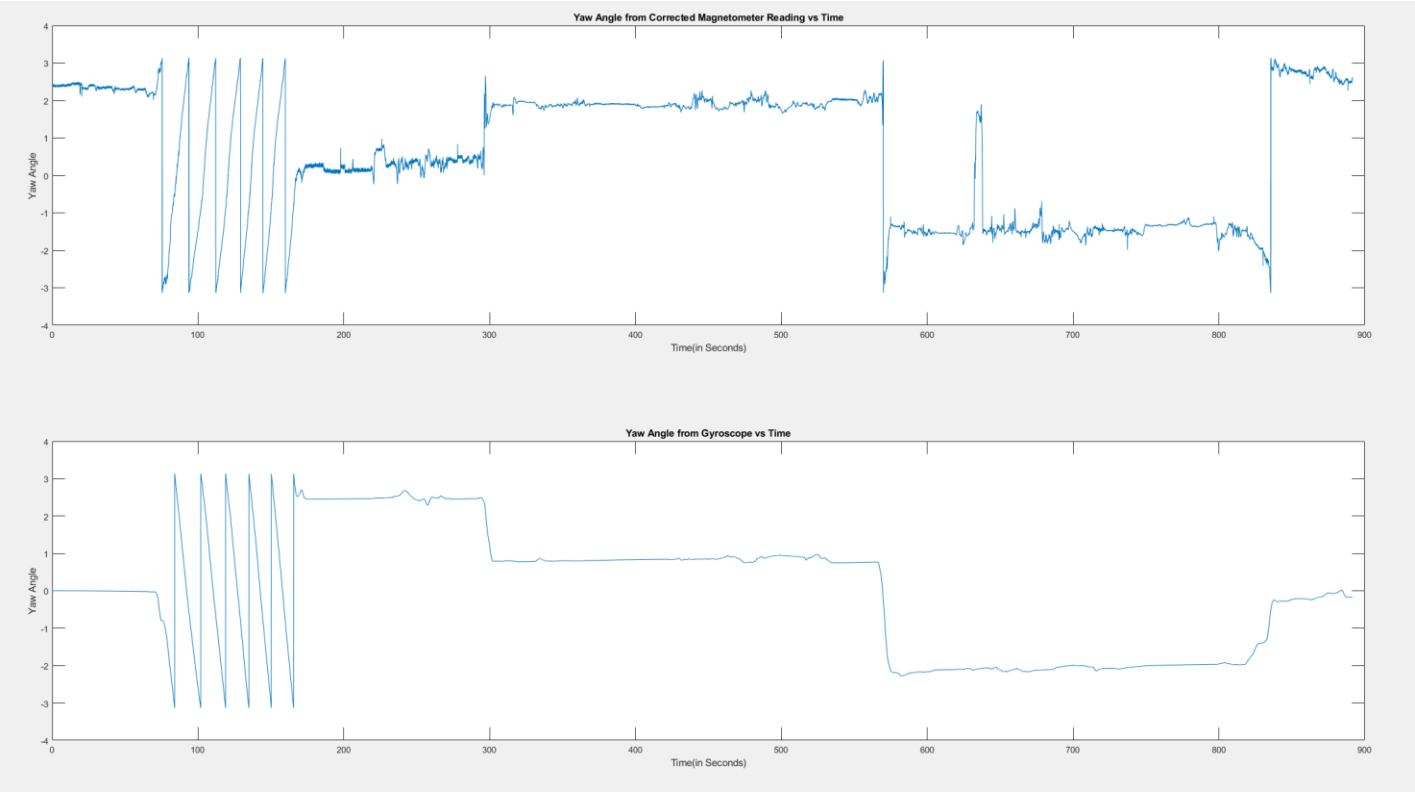
Calculating Yaw Angle from corrected magnetometer readings:

We can do this by just taking the arc tangent of Magnetic Fields along Y and X.

Calculating Yaw Angle from Yaw Rate Sensor readings:

This is done by integrating the Gyroscope around Z value, we use cumtrapz instead of cumsum function to account for the sensor polling rate of 40.

Comparing the Yaw Angle from both methods:



As evident here the yaw angle gyroscope is a lot different than the yaw angle from corrected magnetometer reading which is more accurate. The reason is due to Yaw Drift, single gyroscope has a slight bias in its readings which we haven't addressed, while we integrate the gyroscope value along with its bias, the error also keeps getting worse which is why the different towards the end is massive compared to the initial stages.

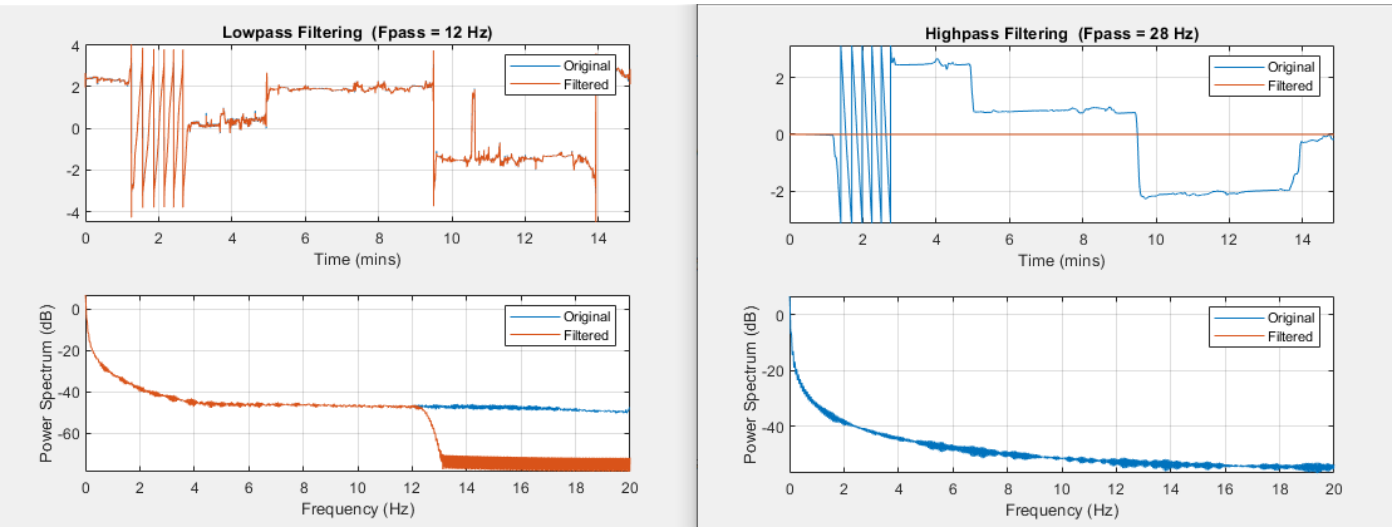
There are also other factors that affect IMU directly such as scale factor (from manufacturing), temperature instability, bias errors, these overtime integrate to bigger errors.

This is where data fusion algorithms come in handy such as Complementary filter which is the next part.

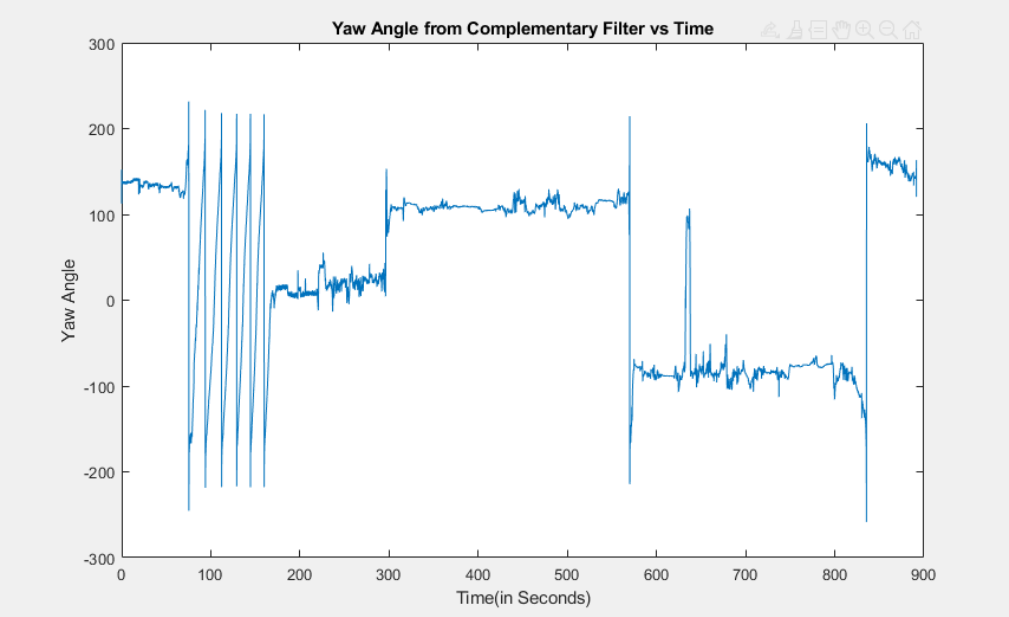
Complementary Filter:

We do this by setting a high pass filter for gyro estimate and a low pass filter for magnetometer estimate, the filter values should be complementary.

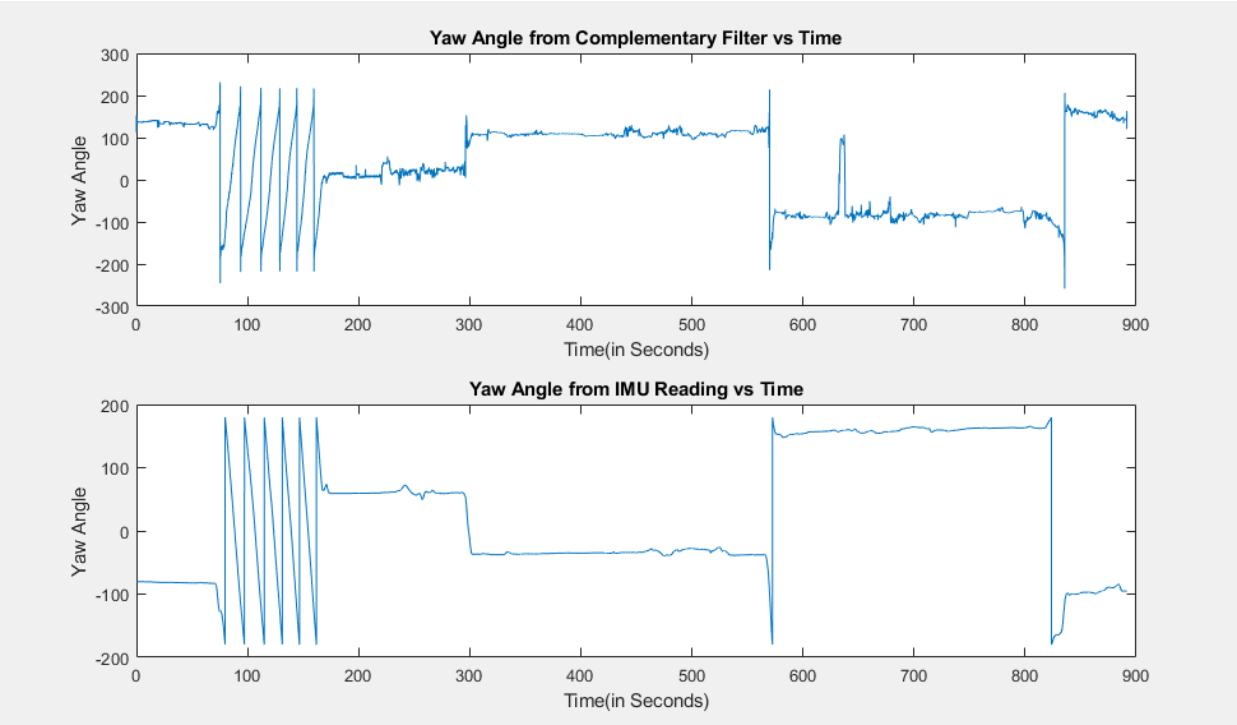
Since the frequency is 40 and if low pass filter = A, then high pass filter = 40 – A.
We've gone with 12 as low pass and 40-12 = 28 as high pass filter in this case.



We take the summation of these 2 filters to create complementary filtered data.



Comparing the Yaw Angle computed by IMU vs Complementary Filtered Yaw Angle:



As expected the Yaw Angle from IMU is a different from the Complementary Filtered one, this is because IMU sensors themselves have a lot accumulated error over time, this could be due to several factors such as:
- Quantization noise: Noise that occurs in the analog-to-digital converter.
- Scale factor error: From manufacturing errors.
- Temperature Instability: Sensor becomes abnormally (more or less) sensitive to inputs.
- Bias error: Zero is not perfect zero.

As the inbuilt calculations use sensor fusion to find parameters, these errors keep getting integrated to a bigger error overtime.

We have various methods to deal with these errors, one of them is Complementary filter as opposed to Extended Klaman filter or Direction Cosine Matrix filter, both Complementary and EKF work in a similar fashion by processing magnetometer, accel and gyro values to yield yaw/pitch/roll, but Complementary filter is simpler with a little slower response time.

Since we dealt with errors in the complementary filter calculation in the preceding steps, the accumulating errors were ‘mostly’ removed leaving very few errors such as IMU surface stability and unstable driving, the differences are evident.

Part2: Estimate the forward velocity

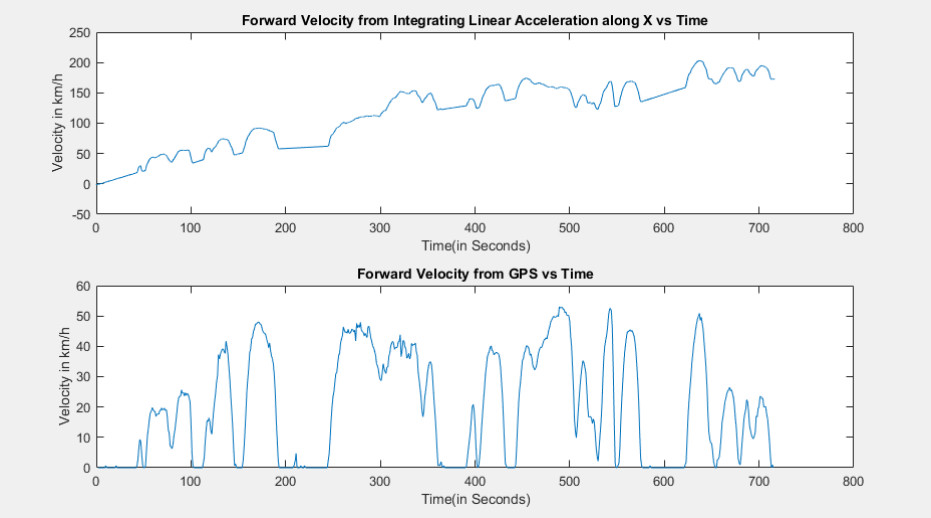
Integrating the forward acceleration to estimate the forward velocity:

Since our IMU was fit in a way that X is forward facing, we integrate Accelerometer reading along X using the cumtrapz function and then multiply it with 3.6 to convert m/s to km/h for more relatable plots.

Calculating an estimate of velocity from GPS measurement:

The idea here is to take subsequent latitude and longitude points and divide by the time in between to find speed and assume it to be velocity at every point.

Plots of the velocity estimates:

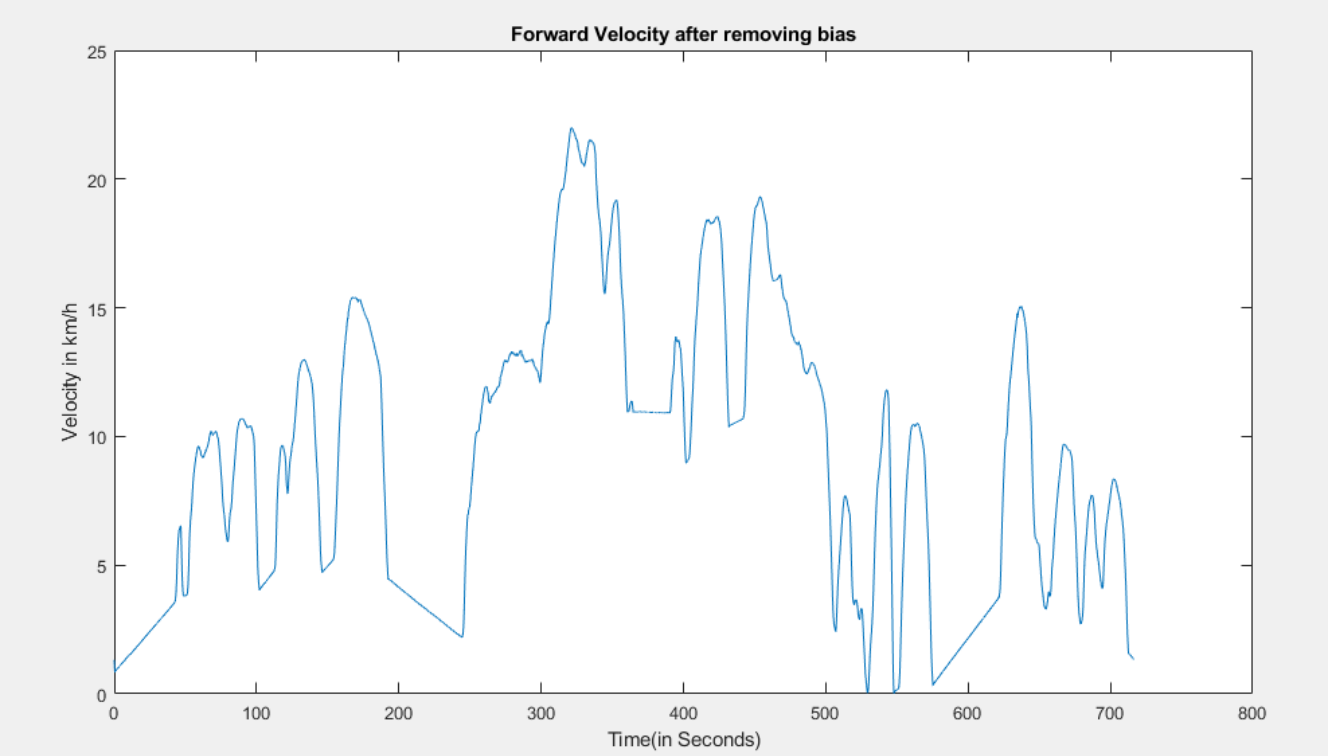


The GPS Velocity plot makes sense, since we did not go past 40 mph which is around 60 kmph, the plot stays under expected limits, the stationary and roundabout part is easily identifiable, although some error is expected since this is a single GNSS received and not a precision GPS system hence each data point accuracy is lower leading to errors in the velocity.

The integrated velocity estimate on the other hand has a lot of errors from previously mentioned IMU errors and the bias also keeps getting integrated which leads to a sort of linearly increasing velocity, the other issue is the negative part which shouldn't happen unless we had issues while taking the data which could be accounted to our TA's rash driving methods, since we had to give priority to safety over smoothness, we had to do sudden brakes and irregular accelerations leading to even more IMU errors during the roundabout part making the velocity part go negative. Hence the data used for this part is from 175s onwards after we exit the roundabout.

Adjustments to the acceleration measurements to make velocity plot more reasonable:

The goal is to make the velocity all positive, scaled down to realistic values and deal with linearly increasing trend, this can be partially cleared by selecting bias from specific time frame and subtracting it throughout the whole data. For this, from trial and error, the bias from 170 -> 210 was selected and removed by subtracting the mean from it throughout the data and to deal with the negative values, min value was taken and subtracted throughout the data to push it up slightly to keep all the readings positive.

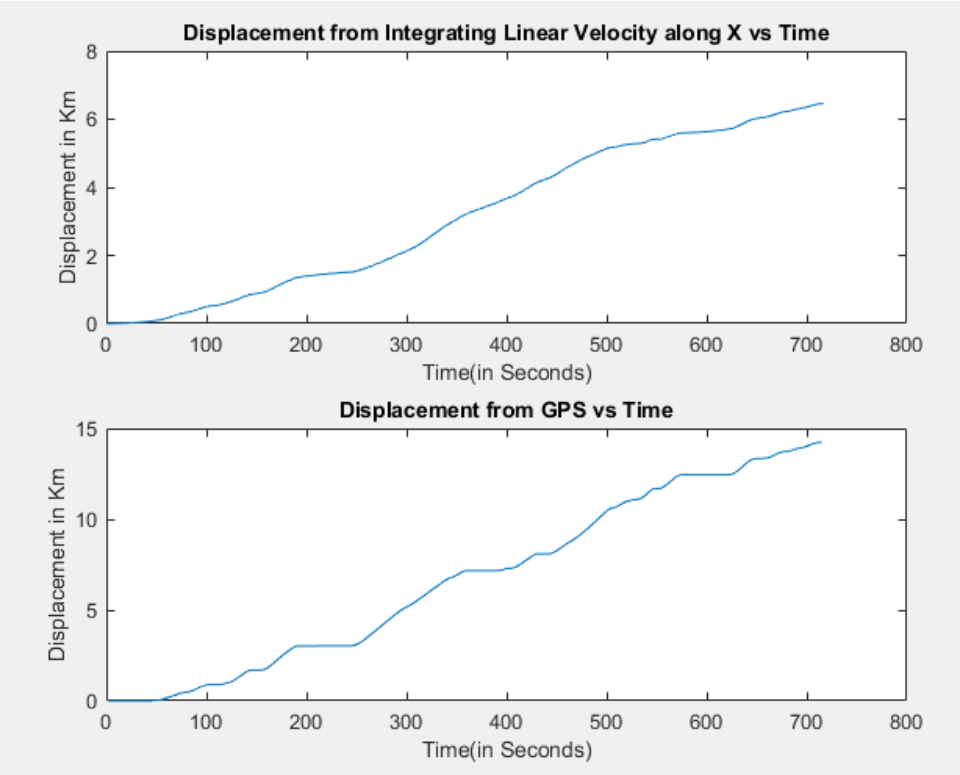


The scaling is still not appropriate which means the bias removal is not perfect and hence the speeds are much lower than realistic and GPS values.

Part3: Dead Reckoning with IMU:

Integrating IMU data to obtain displacement and compare with GPS:

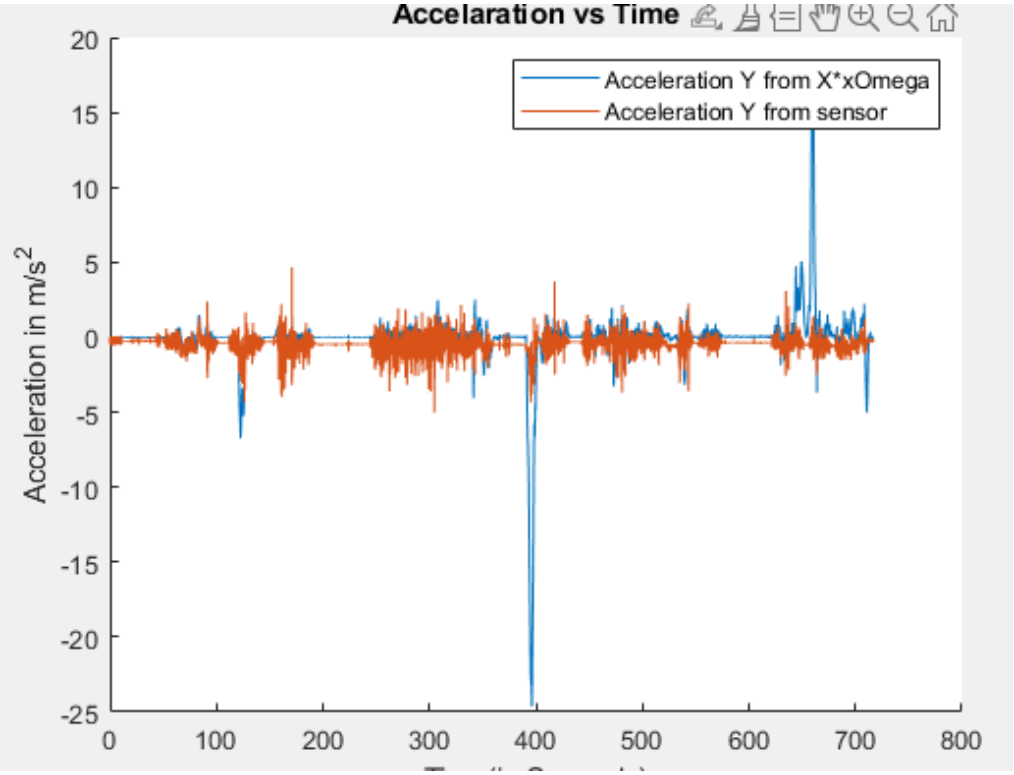
We integrate the GPS and IMU velocity from previous part here to obtain the displacements.



The displacement from GPS is a lot more accurate here and shows varying speeds that correspond to the irregular, non-smooth displacements. Whereas displacement from the corrected IMU data is a lot more smooth but incorrect because we could not completely remove the biases and other errors in the previous case while obtaining the velocity from integrating acceleration. This means that the errors will keep compounding when we integrate it once and then again for displacement hence leading to a smooth more than linearly increasing plot.

Dead Reckoning: Part 1

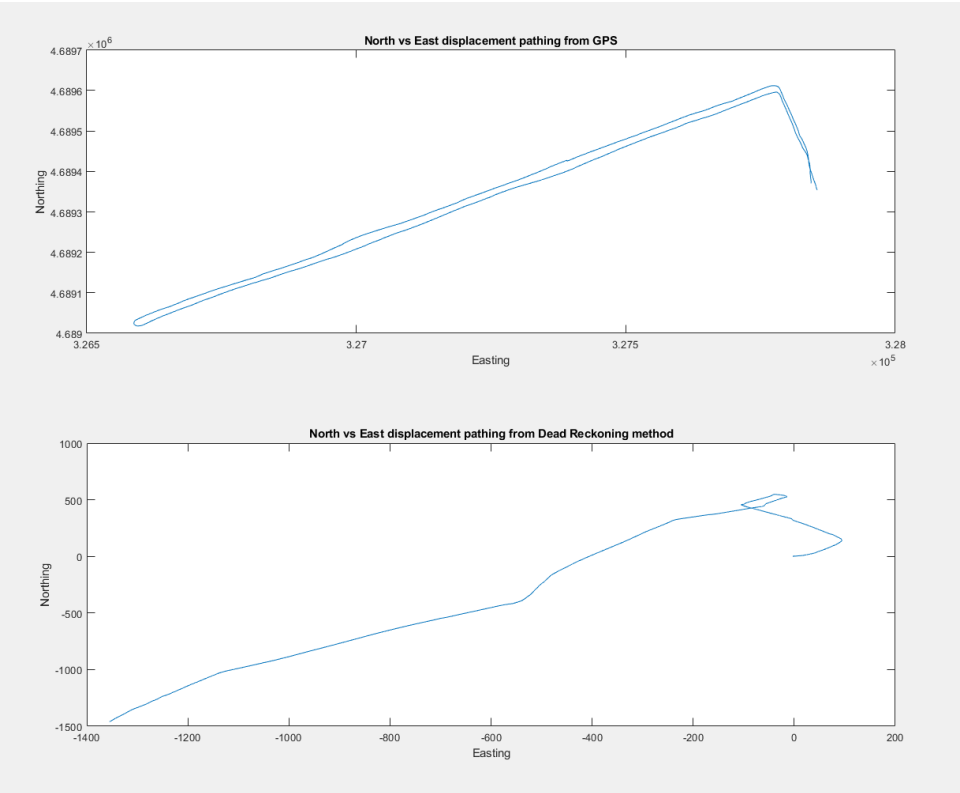
Acceleration from product of Yaw Angle and Filtered Velocity



The acceleration from product on yaw rate and forward velocity seems to be a scaled-up version of the acceleration along y from the IMU sensor directly.

One possible reason is because of the yaw rate itself, Gyroscopes are subject to bias instabilities, which means the initial zero will drift overtime due to integration of inherent imperfections and noise within the device, this means that the yaw rate strays further away from accurate reading as time progresses. This is evident from the graph since the blues and reds are further apart as time progresses which means the accumulated errors keep increasing and get multiplied with Acceleration along X leading to all the variations.

Dead Reckoning: Part 2



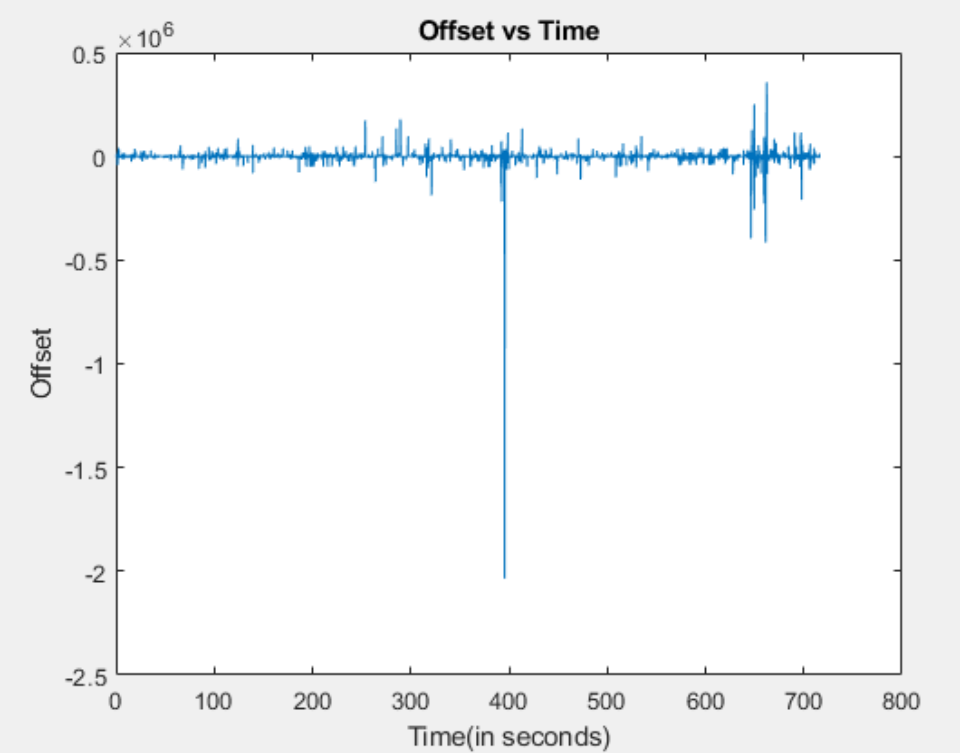
We take the complementary filtered yaw angle and filtered acceleration (bias removal) from 175s onwards and find the easting and northing by taking the cos and sin components of the acceleration w.r.t filtered yaw angle.

There are some similarities between the paths, they are vaguely heading in the same direction, but Dead Reckoning method seems a lot more disoriented which is due to sensor fusion, we've been incompletely removing biases and imperfections in our previous cases which leads up to all the remaining errors being integrated twice.

Our Yaw angle involved readings from magnetometer, which could've been affected by the compass and other magnetic fields in the nuance car, soft-iron and hard-iron corrections most probably were not all the errors present, other factor could be due to non-smooth driving since we had a lot of interruptions, signals during our ride which would've jerked the car causing the IMU to not work as intended.

Dead Reckoning: Part 3 Estimating Xc

This is done by simply subtracting X*xOmega from Y**obs from part 1 and divide it by Omega*



And the mean Xc is(after removing outliers/infinites to avoid NaN mean):

```
MeanXcc =

 -162.9863
```

Conclusion:

Sensor fusion can be very handy to find parameters by methods such as dead reckoning which will come in handy in navigating through enclosed structures where GPS signals may not reach, but it is also important to remove all possible errors and biases from the sensors as they accumulate overtime giving highly inaccurate navigation.