EECE5554 Robotics Sensing and Navigation

Lab 4: Navigation with IMU and Magnetometer

Analysis Report By Aryaman Shardul Lnu

I will be uploading the drivers to my GitLab Folder for Team 14. My GitLab username is: lnu.arya

1. Introduction

The lab had two main data collection parts for analysis: the car donuts (going around in circles) and the mini Boston tour (collecting the moving data on Boston Streets). We used the NUANCE car for this. I have analyzed the data obtained from the GPS and the VN-100 IMU below that was published by our "gps_driver.py" and "imu_driver.py". There are a few important things to note. I have published the angular X, Y, and Z velocities in rad/s as given by the VN-100 and required by the "sensor_msgs/Imu.msg". But for the ease of visualization, I have converted them to deg/s for the analysis part. Similarly, the data for the X, Y, and Z magnetic fields is published in Tesla (T) as required by the "sensor_msgs/MagneticField". But for convenience, I have converted them to Micro Tesla (μT) for analysis at certain places. Also, the Quaternions published by the driver have been converted into Euler angles for the analysis utilizing the formulae:

$$yaw = atan2(2(q_w * q_z + q_x * q_y), 1 - 2(q_y^2 + q_z^2)) \quad (1)$$

$$pitch = asin(2(q_w * q_y - q_z * q_x)) \quad (2)$$

$$roll = atan2(2(q_w * q_x + q_y * q_z), 1 - 2(q_x^2 + q_y^2)) \quad (3)$$

The above equations give the Euler angles in radians. I have further converted them into degrees for ease of analysis.

2. Estimating the heading (Yaw)

2.1. Magnetometer Calibration

Figure 1 shows the magnetometer X-Y plot before and after hard and soft iron calibration, respectively. The "data_going_in_circles" was used for plotting these graphs.

To calibrate the magnetometer data collected, I performed corrections for both hard-iron and soft-iron distortions. First, I computed the mean of the raw X and Y magnetic field values, which represent any constant offset due to hard-iron distortion. By subtracting these mean values from the magnetometer readings, I tried to center the data around the origin. This removes the hard-iron offset. Next, I applied Singular Value Decomposition (SVD) to the centered data, which helped approximate the best-fit ellipse around the points. Using SVD, I analyzed the distribution of magnetometer data points by decomposing them into components that represent the best-fit ellipse. I then applied a transformation matrix derived from the SVD to a predefined unit circle, shaping it into an ellipse that matches the data distribution. The SVD provided the ellipse's rotation angle (theta) and the lengths of the semi-major and semi-minor axes (a and b), representing the soft-iron distortions. Using these, I performed a transformation to align the data along the principal axes of the ellipse. I first rotated it by theta to correct for misalignment and finally scaled it to convert the ellipse into a circle by applying the appropriate scaling factors to correct for axis distortion.

The different sources of distortion that were present were the presence of laptops and mobile phones in the car as well as other electronic devices. The car's battery and potentially it's structural components could have caused the hard iron defects. The car's engine, metal frame, and other electronic systems in it could have caused the soft iron defects. Also, magnetic fields generated from other buildings and vehicles caused some distortions. Finally, the moving traffic also generates some distortion.

The presence of a consistent offset in the magnetometer data where the plot was shifted away from the origin indicated a hard-iron effect. Hard-iron distortions typically cause a uniform shift in all magnetometer readings, moving the data cloud away from the origin. The magnetometer plot showed an elliptical rather than a circular shape, which pointed to soft-iron distortion. Soft-iron effects stretch or compress the data along certain axes due to interference from nearby materials like the car's metal

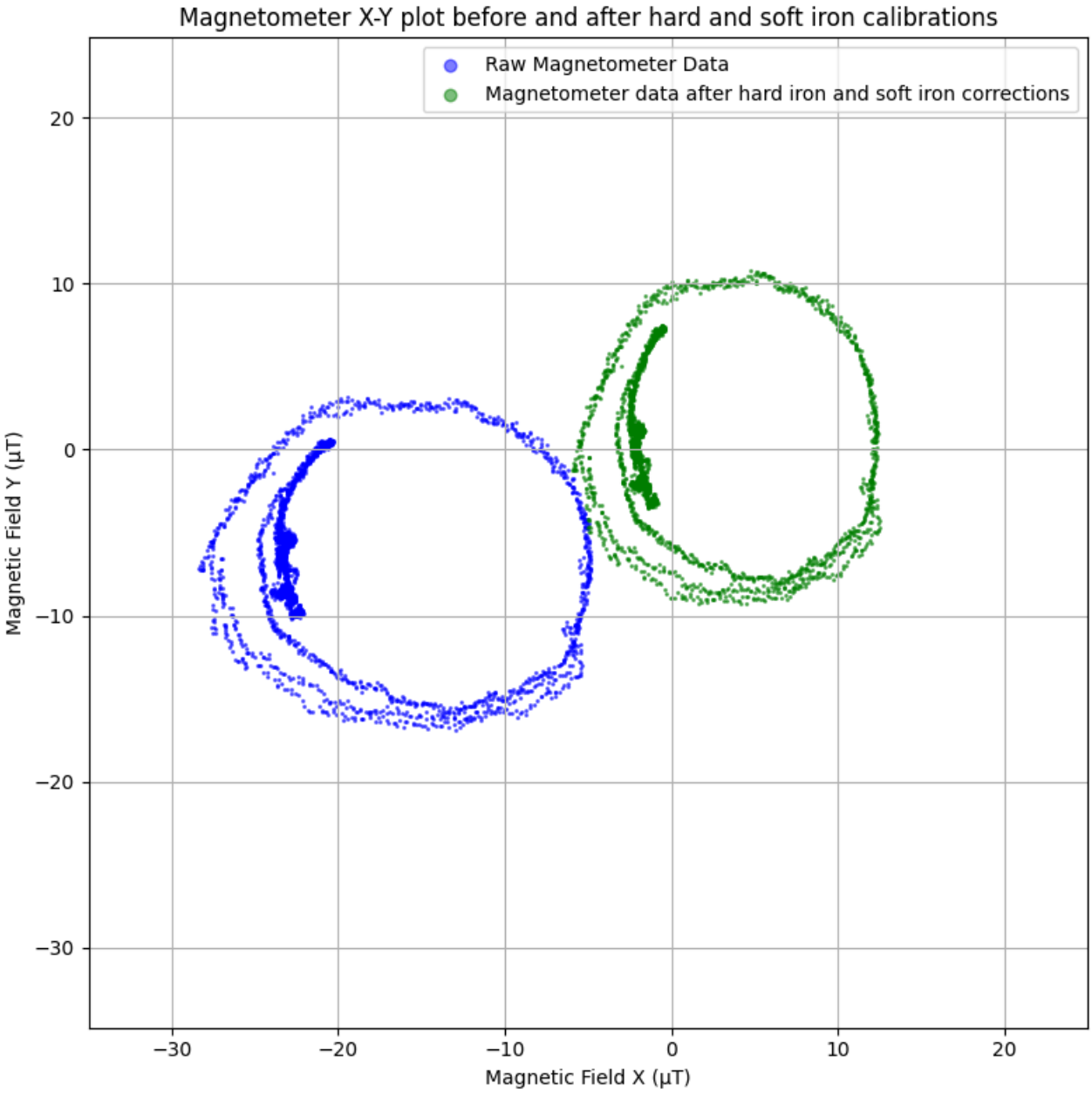

Figure 1: Magnetometer X-Y plot before and after hard and soft iron calibrations

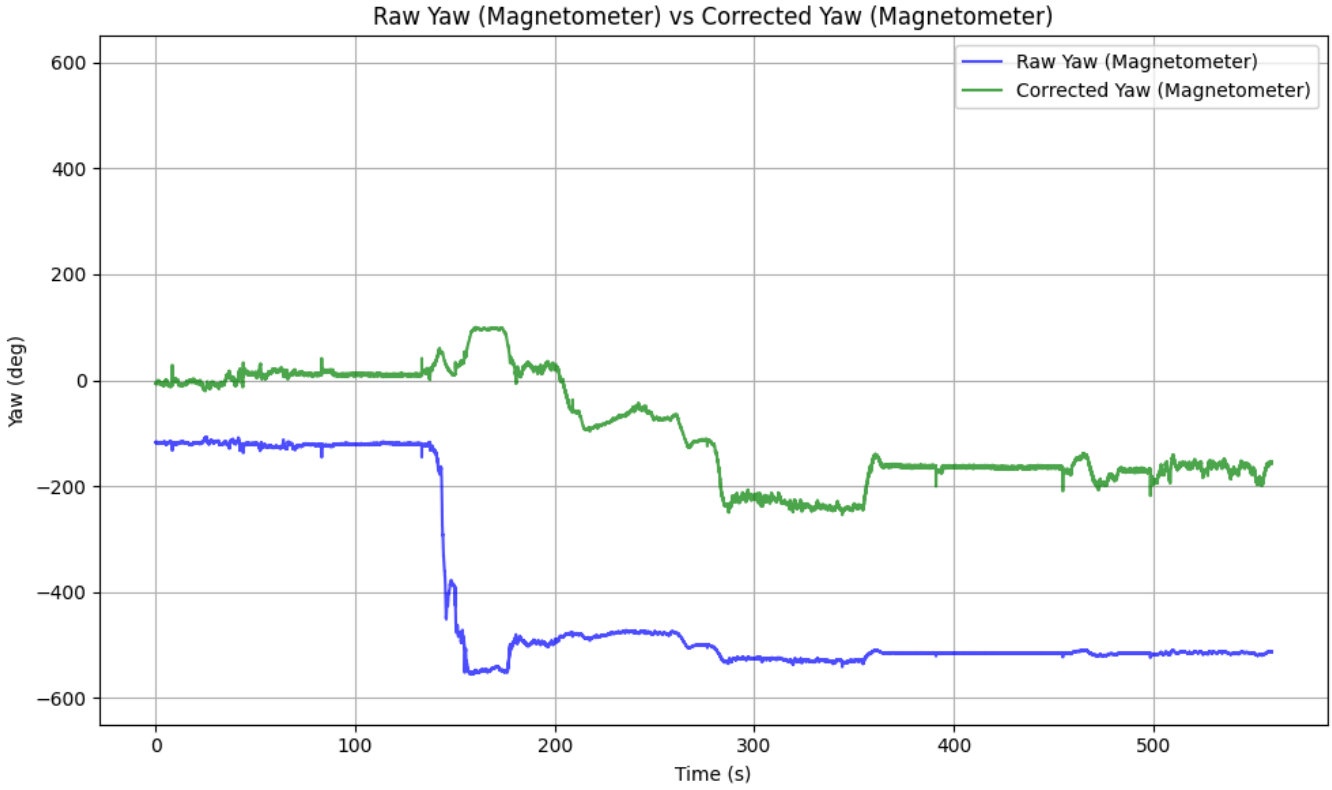

Figure 2: Time series magnetometer data (Yaw) before and after the correction

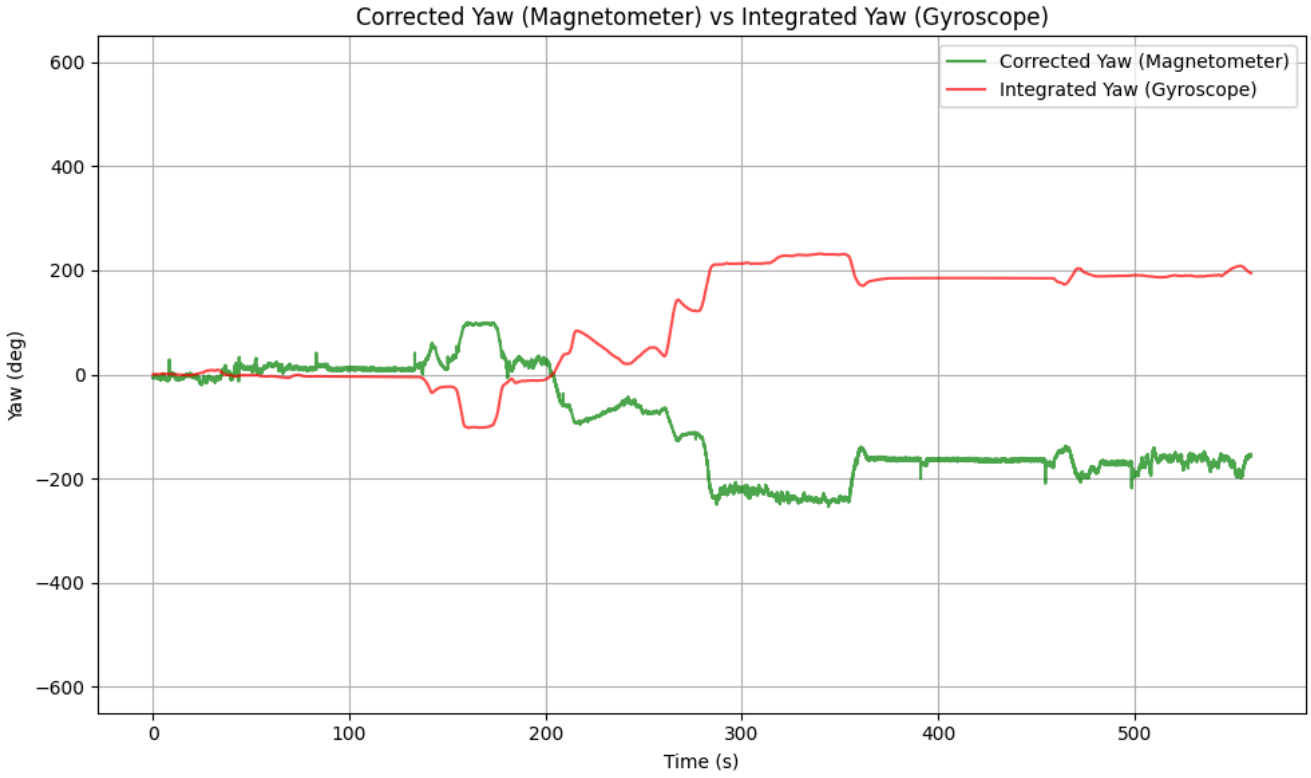

Figure 3: Magnetometer Yaw & Yaw Integrated from Gyroscope

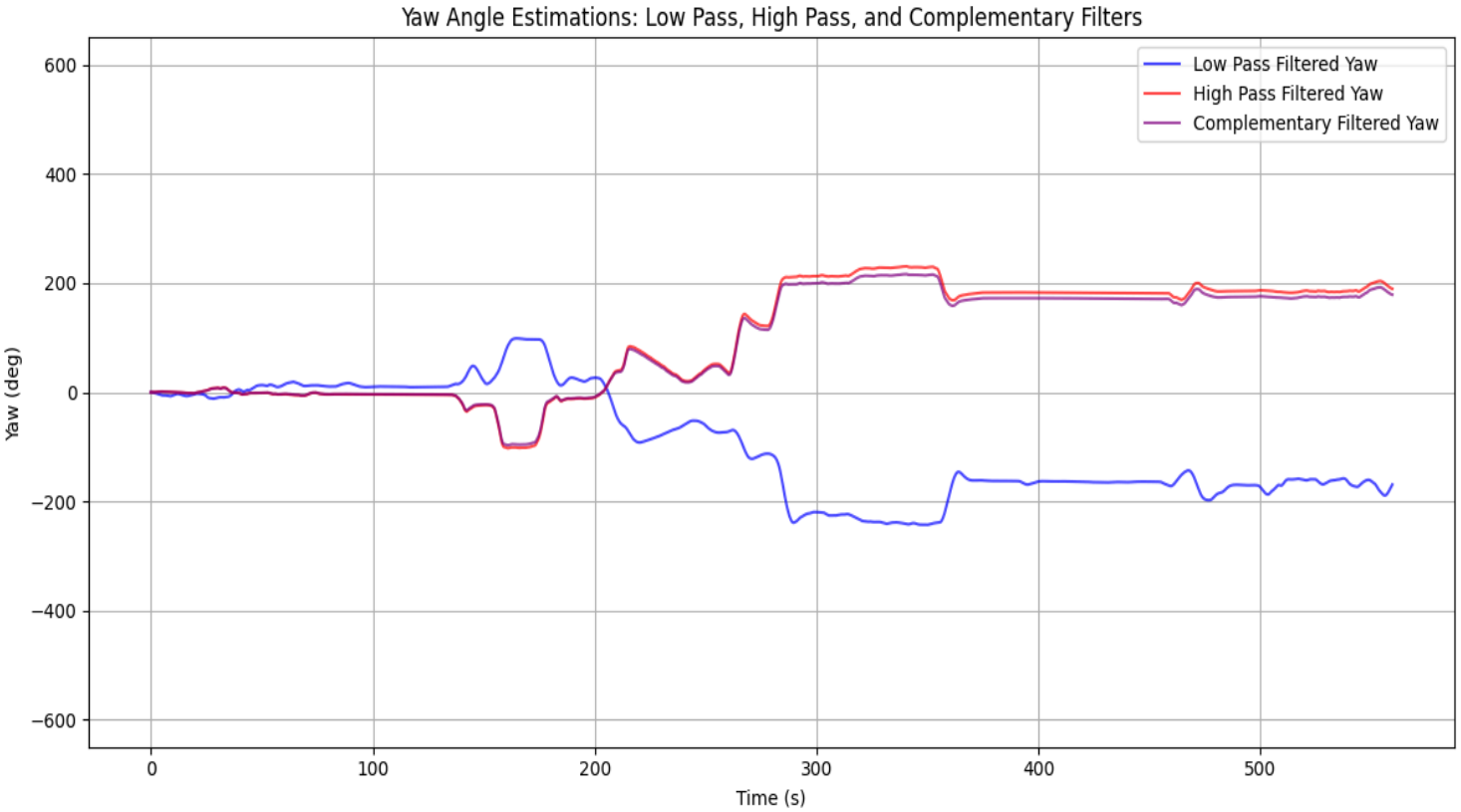

Figure 4: Low Pass (On Corrected Magnetometer Data), High Pass (On Integrated Gyroscope Data), and Complementary Filters for Yaw angle estimation

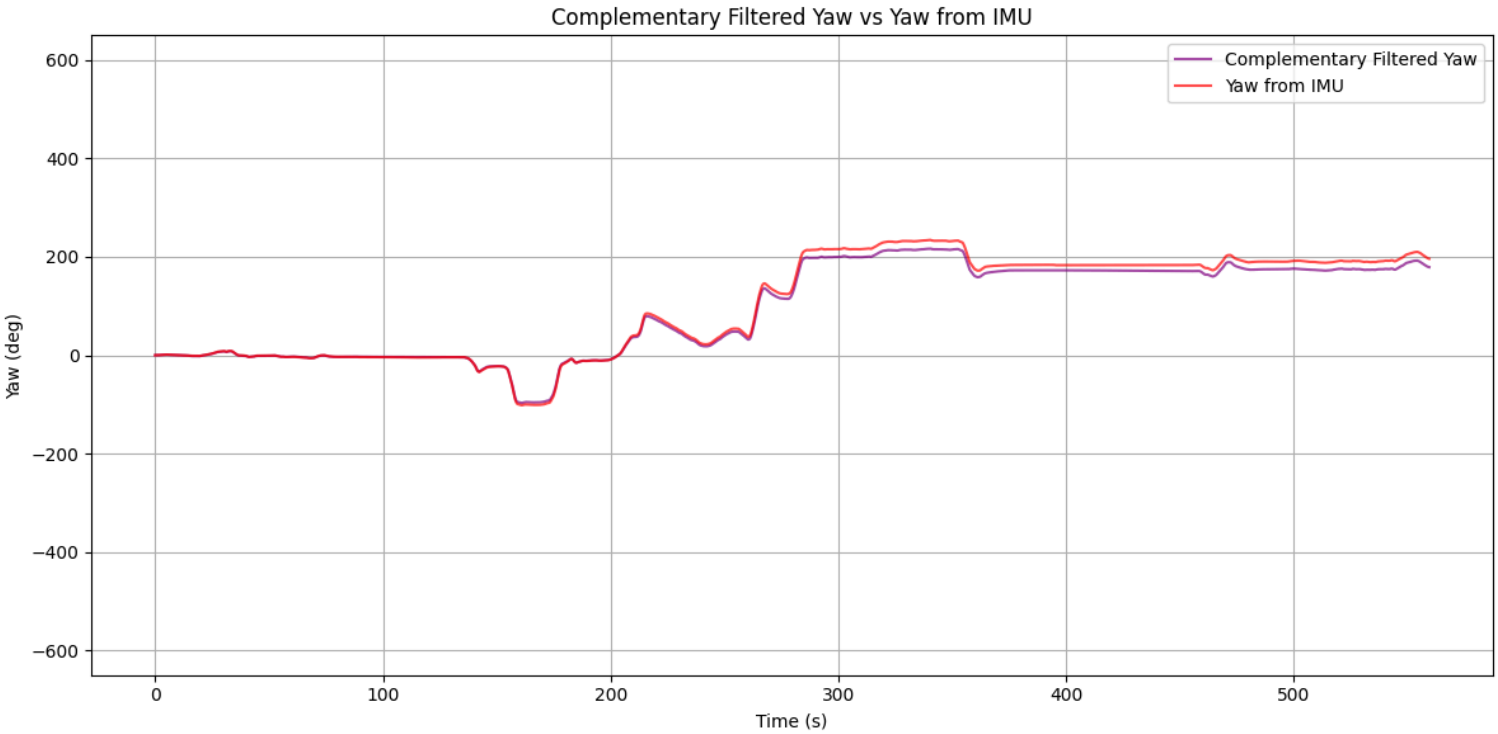

Figure 5: Yaw from Complementary Filter vs Yaw provided by te VN-100 IMU (Obtained from quaternions)

chassis. I noticed occasional fluctuations and irregularities in the magnetometer readings that did not match the expected gradual changes based on the on car's movement. These spikes suggested some electromagnetic interference from nearby electronics in the car, as electromagnetic noise can create abrupt changes in the magnetic field readings.

2.2. Sensor Fusion

From here onward, all the graphs are plotted using the "data_driving" dataset.

I used the same magnetometer calibration to remove the hard and soft iron defects from this dataset too. After that, I plotted a graph of the raw yaw angle and the corrected yaw angle obtained from the magnetometer, as shown in Figure 2. Post that, I integrated the Z-axis of the gyroscope using the "cumtrapz" function to get the yaw angle from the gyroscope. I went on to compare this yaw with the yaw obtained from the corrected magnetometer readings. This is depicted in Figure 3.

The yaw from the magnetometer has large noises (high-frequency characteristic), while the yaw from Gyroscope has bias/drift (low-frequency characteristic) that causes errors in the yaw to increase with time. To remove these errors, I used a low-pass filter on the magnetometer data and a high-pass filter on the Gyroscope data. I used a "Butterworth" filter to implement these two filters. For the low-pass filter on magnetometer data, I set the sampling frequency to 40 Hz and the cut-off frequency to 0.09 Hz. I applied a high-pass filter on the gyroscope with the same sampling frequency of 40 Hz but a low cutoff frequency of 0.00001 Hz. To fuse the two filtered signals, I weighted the high-pass filtered gyroscope data heavily with a factor (α) of 0.97, favoring the gyroscope's responsiveness, and combined it with the low-pass filtered magnetometer yaw data using the equation given below:

$$CF = (1 - \alpha) * LF + \alpha * HF \tag{4}$$

where CF stands for complementary filter, LF stands for Low-Pass filter that represents the magnetometer yaw data, and HF stands for High-Pass filter that represents the yaw obtained by integrating the gyroscope data. This weighting allowed the complementary filter to dynamically adjust the yaw estimate based on the strengths of each sensor, with the gyroscope dominating in dynamic situations and the magnetometer stabilizing the yaw in static or low-movement scenarios. I have also used the "unwrap" function to analyze the difference between consecutive yaw angles and adjust the phase angle by adding or subtracting 2π radians to ensure continuity if the phase angles wrap around at -π or π radians. The graphs of the low-pass filter, high-pass filter, and complementary filter can all be seen together in Figure 4. I have also plotted the yaw obtained from the complementary filter along with the yaw obtained from the IMU sensor in Figure 5 obtained by converting quaternions to yaw using the formula:

$$\begin{aligned} t1 &= 2.0 * (qw * qz + qx * qy) \\ t2 &= 1.0 - 2.0 * (qy * qy + qz * qz) \\ yaw &= arctan2(t1, t2) \end{aligned} \tag{5}$$

where qx, qy, qz, and qw represent the four quaternions.

I would trust the estimates for yaw provided by the complementary filter more than the corrected magnetometer yaw. This is because the complementary filter effectively balances the advantages of both sources, allowing it to provide a reliable estimate that remains stable during motion. Also, by reviewing the graphs of both the complementary filter and the magnetometer yaw, the graph of complementary filter is quite closer to the yaw provided by the Imu as compared to the magnetometer yaw that is filled with noise.

3. Estimating the Forward Velocity

For this section, I first integrated the forward acceleration obtained from the accelerometer to get the forward velocity. Also, I found the GPS's forward velocity using the formula,

$$\begin{aligned} Vel\_Easting &= \Delta E / \Delta t \\ Vel\_Northing &= \Delta N / \Delta t \\ Forward\_Vel &= \sqrt{Vel\_Easting^2 + Vel\_Northing^2} \end{aligned} \tag{6}$$

where Δ_E = Difference in consecutive UTM easting co-ordinates, Δ_N = Difference in consecutive UTM northing co-ordinates, and Δ_t = Difference in consecutive GPS timestamps.

I plotted these two forward velocities in Figure 6. As we can see, the forward velocity obtained by the GPS is relatively stable, whereas the one obtained by integrating the accelerometer's data

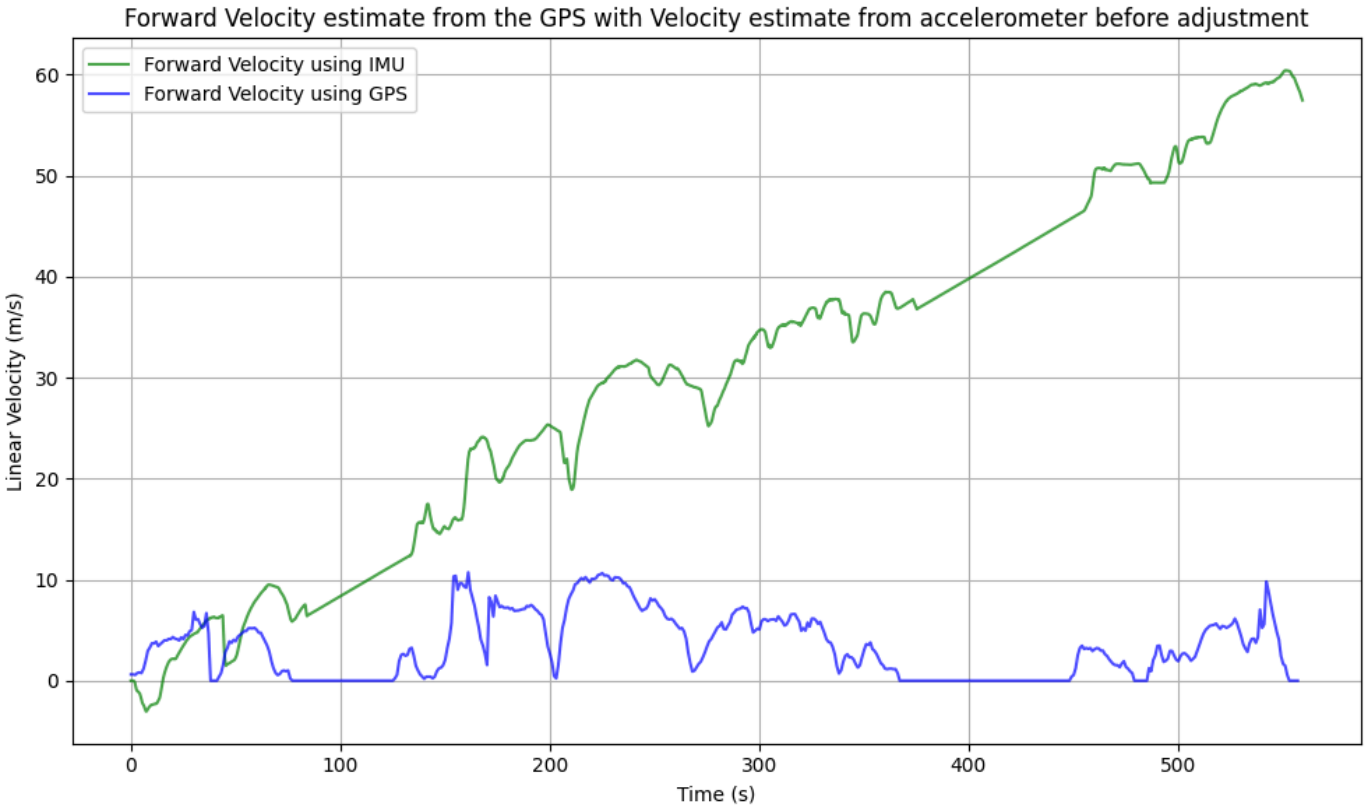

Figure 6: Forward Velocity estimate from the GPS with Velocity estimate from accelerometer before adjustment

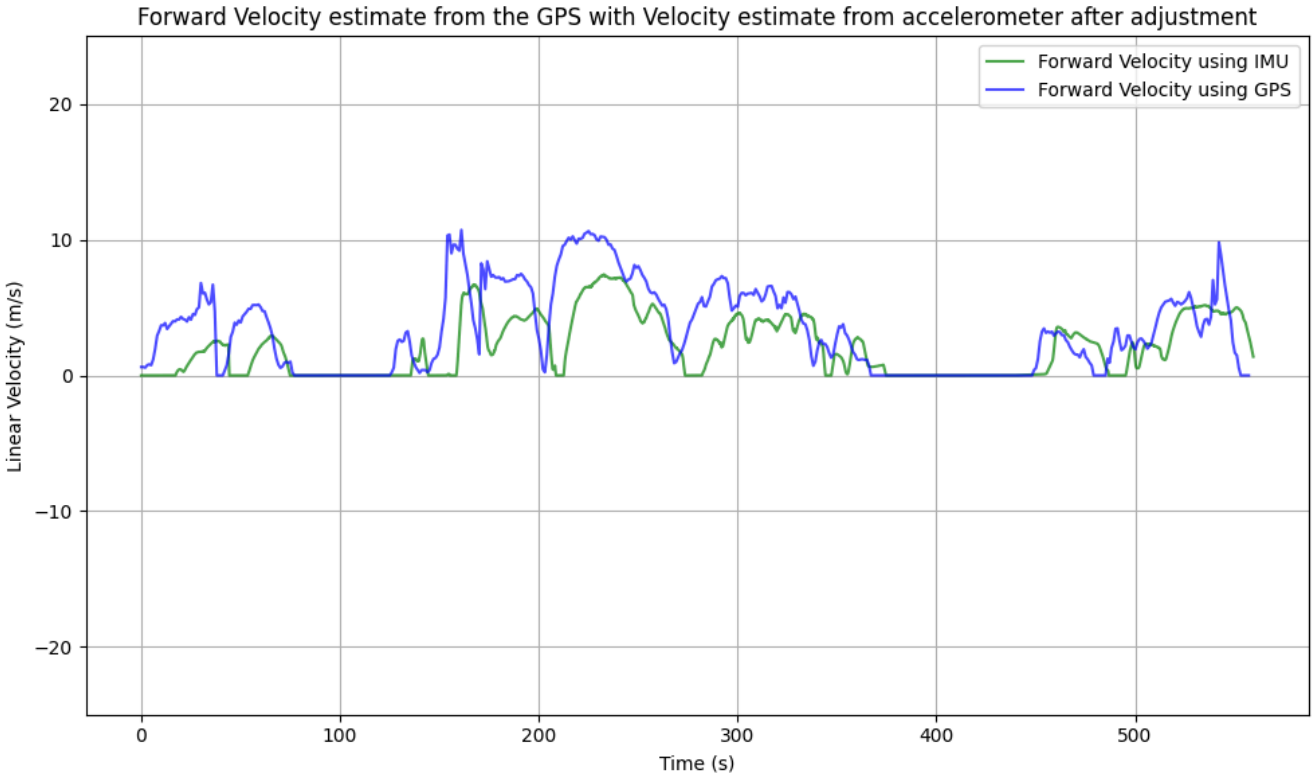

Figure 7: Forward Velocity estimate from the GPS with Velocity estimate from accelerometer after adjustment

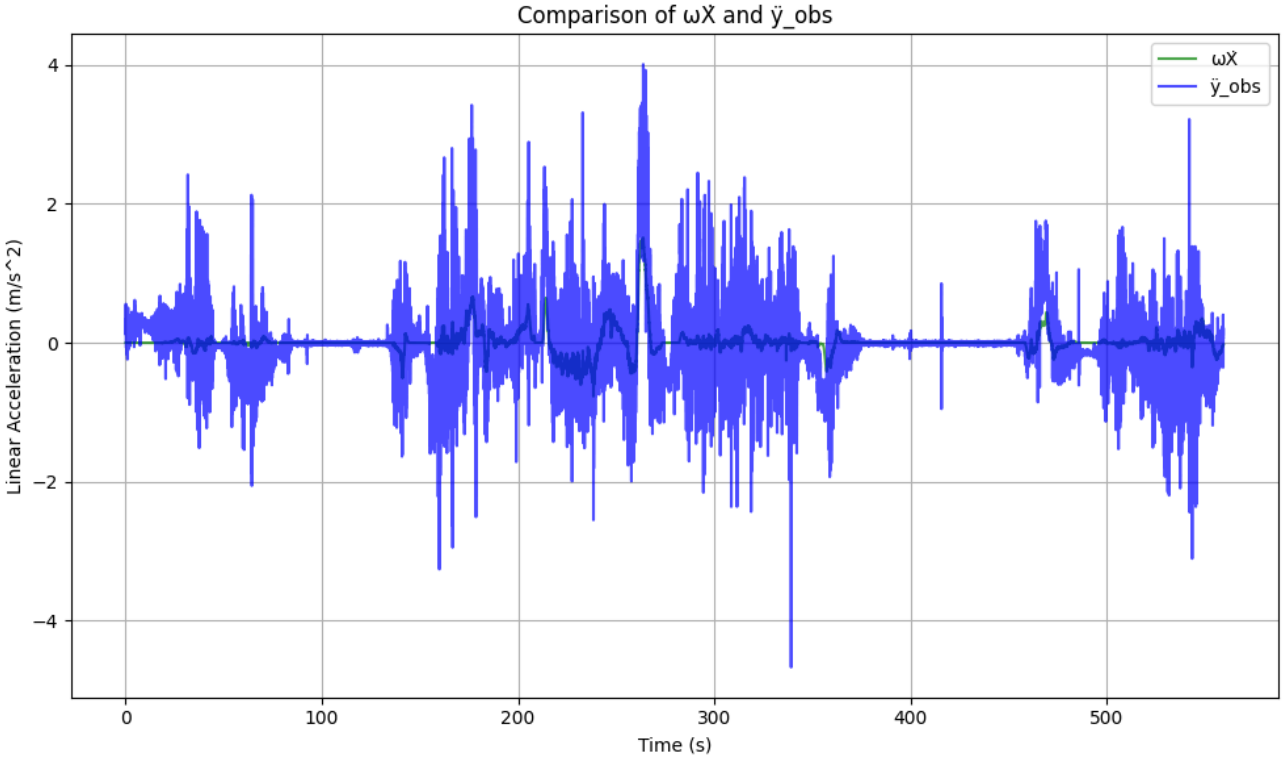

Figure 8: Comparison between $\omega\dot{X}$ and $\ddot{y}\_obs$

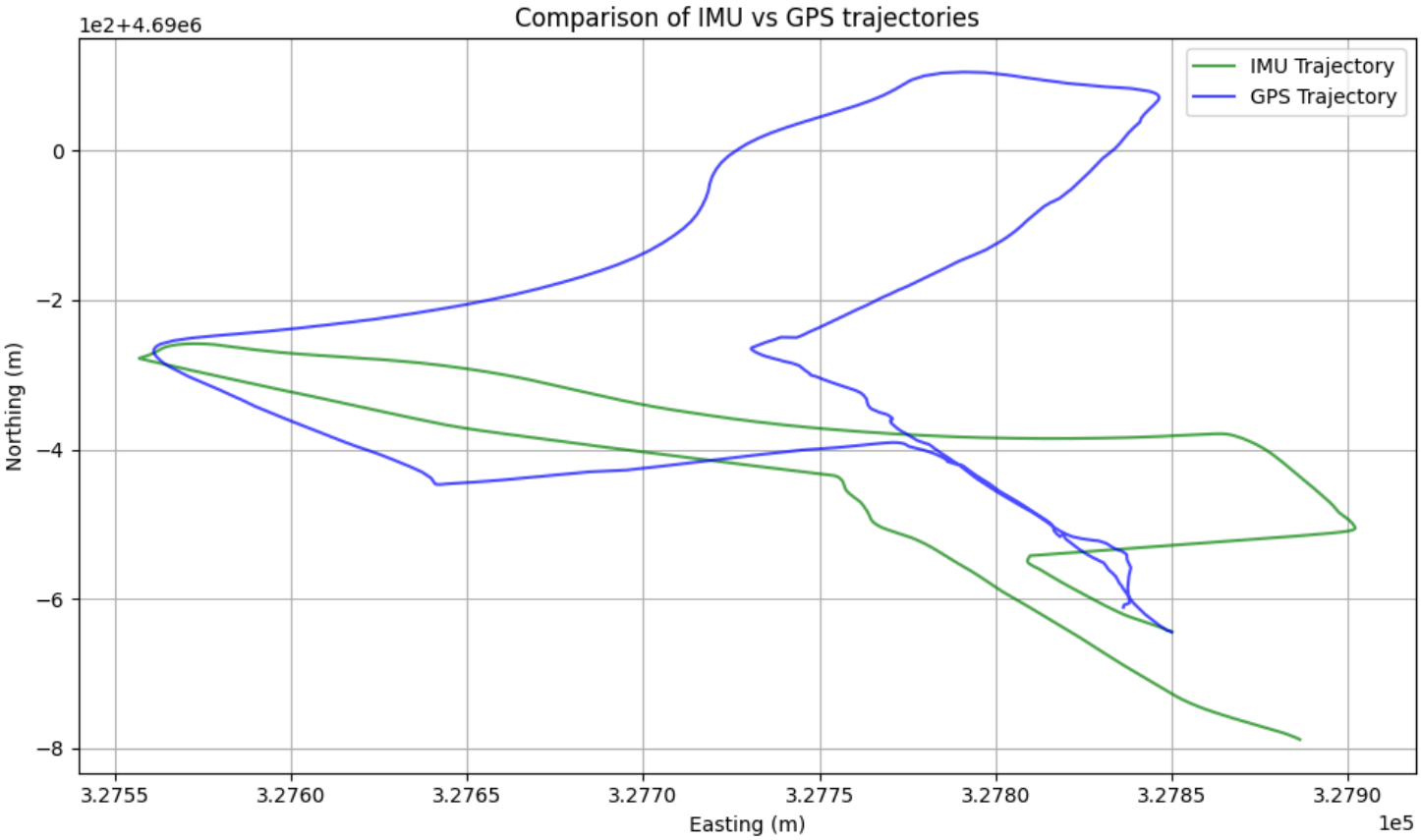

Figure 9: Comparison of IMU and GPS trajectories

is constantly increasing with time. This is mainly due to the bias/offsets generated by the stationary points where the acceleration's value should be zero, but it is not, and hence, integrating such errors has led to a huge increase in the forward velocity's value.

To fix the discrepancies in the forward velocity obtained from the accelerometer's data, I first removed the initial bias by calculating the mean acceleration value of the accelerometer's data and then subtracted it from each value in that dataset. Post that, I created a function that detects points below a certain threshold and I assume them to be stationary and give them the value zero for acceleration. But, since those values were below the threshold but actually not zero, they form the offset, which needs to be subtracted from each acceleration data point during the phase in which the car moves right until we encounter the next set of stationary points where we can recompute the new offset value for the further points. The graph of the corrected forward velocity for the accelerometer data and the forward velocity from the GPS data have been shown in Figure 7.

The main discrepancies that are present in the velocity estimate between the accelerometer and GPS are the drifting of the velocity obtained from the accelerometer due to it being an integrated value (all the small errors add up to form a huge drift) and the lag in capturing rapid speed changes by the GPS due to its lower sampling frequency and possible latency as compared to the accelerometer due to signal processing, leading to discrepancies in dynamic conditions.

4. Dead Reckoning with IMU

First, I calculated the product of gyro z and the IMU's forward velocity ($\omega\dot{X}$) and compared it with accel y ($\ddot{y}_{obs}$) as depicted in Figure 8. Note, only for this graph's calculations, I have used "rad/sec" as the units for ω. Both $\omega\dot{X}$ and accel y ($\ddot{y}_{obs}$) show similar patterns, with peaks and valleys that align relatively well. This indicates that the overall trend or response of the system in terms of acceleration is consistent between the two estimates. But there are noticeable differences in the magnitude, especially in sections where $\ddot{y}_{obs}$ exhibits higher spikes (both positive and negative) than $\omega\dot{X}$. This is due to the assumption that $\dot{Y}$ (no sideways skidding) and ignoring the offset (x_c = 0). In reality, minor skidding or offset effects could affect the sensor readings, leading to discrepancies. Also, due to ignoring any sideways skidding, I did not account for the effect of stationary points on the accelerometer's Y axis.

After that, I defined the vector v_imu as (v_e, v_n) where,

$$\begin{aligned} v_e &= v\_imu * cos(cf\_yaw) \\ v_n &= v\_imu * sin(cf\_yaw) \end{aligned} \tag{7}$$

Here, v_imu represents the forward velocity obtained by integrating the forward acceleration, v_e is the easting component of v_imu and v_n is the northing component of v_imu. cf_yaw represents the complementary filter yaw angle which is in radians for this formula.

After this, I integrated v_e and v_n to obtain x_e and x_n, i.e., the easting and northing components used to estimate the trajectory of the vehicle. I have plotted the trajectory of the vehicle estimated using the IMU and the trajectory of the vehicle obtained using the GPS co-ordinates in Figure 9. As you can see, the GPS trajectory shown in blue represents the path we took quite well. The IMU trajectory doesn't exactly match the GPS's trajectory, although we can see that it's overall shape is quite similar. This could maybe happen due to some noise in the heading angle (yaw) that has not been filtered or detected yet. I calculated a scaling factor to compare the GPS and IMU trajectories. First, segment distances were computed separately for both GPS and IMU data using the differences in UTM easting and northing coordinates. The scaling factor was then determined by taking the mean ratio of IMU segment distances to the corresponding GPS segment distances. This factor represents the average proportional relationship between the distances measured by the IMU and the GPS.

Given the specifications of the VectorNav, the VN-100 could potentially navigate without a position fix for about 1–3 minutes before position error becomes substantial. This duration can vary based on environmental factors, calibration quality, and filtering techniques, but for practical applications, it's challenging for the IMU alone to navigate accurately without position corrections for extended periods. The GPS and IMU paths didn't exactly match maybe because of inaccuracies in the yaw data.