STM32-Based IMU Signal Processing with Python-Aided Visualization and Fusion

Nabil KANA

Overview

The objective of this project is to process raw data from a 6-axis IMU (MPU-6500), extract orientation information (roll, pitch, yaw), and reduce noise and drift using sensor fusion techniques. This involves building drivers on STM32, filtering, and applying both complementary and Kalman filters to estimate reliable orientation data. The project highlights limitations such as gyroscope drift, accelerometer noise, and phenomena like gimbal lock at high pitch angles.

Driver architecture

The driver starts by trying to wake up the device by sending a wake up command 0x00 to power management register 0x6B , the MPU address is defined as 0x68<<1 if AD0 is ground else 0x69<<1 . in order to make sure our device is up we send Who_am_i command 0x00 to the who am i register , if none of the two calls return 0 , our wake up function should return a True which we will use in our main code to try multiple calls .

The slave address of the MPU-6500 is b110100X which is 7 bits long. The LSB bit of the 7 bit address is determined by the logic level on pin AD0. This allows two MPU-6500s to be connected to the same I^2C bus. When used in this configuration, the address of the one of the devices should be b1101000 (pin AD0 is logic low) and the address of the other should be b1101001 (pin AD0 is logic high).

-MPU Datasheet - page 28-

Two separate functions were defined to read **accelerometer** and **gyroscope** data from the MPU-6500. The sensor transmits each axis measurement in **two bytes** To reconstruct the full signed 16-bit value, a helper function is used `static int16_t combine_bytes(uint8_t high, uint8_t low)` .This function combines the two bytes and returns a signed 16-bit integer. The resulting values are stored in `two separate structs passed by reference from main code` ,the data is read from two registers dedicated to accelerometer and gyroscope defined in header file MPU9250_ACCEL_XOUT_H 0x3B and MPU9250_GYRO_XOUT_H 0x43 ,By default, the **accelerometer** is configured to ±2g full-scale sensitivity. Since the maximum positive value of a signed 16-bit integer is `0x7FFF = 32767`, and the measurement range is 4g total (from -2g to +2g), the resolution becomes:32768/2=16384 LSB/g , the gyroscope default sensitivity is ±250°/s which gives us 32768/250=131 . Theses conversions are implemented in **`MPU9250_ReadAccel and MPU9250_ReadGyro`**. The sensitivity list can be found in the register Map and description in page 14.

3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]
41	65	TEMP_OUT_H	R	TEMP_OUT_H[15:8]
42	66	TEMP_OUT_L	R	TEMP_OUT_L[7:0]
43	67	GYRO_XOUT_H	R	GYRO_XOUT_H[15:8]
44	68	GYRO_XOUT_L	R	GYRO_XOUT_L[7:0]
45	69	GYRO_YOUT_H	R	GYRO_YOUT_H[15:8]
46	70	GYRO_YOUT_L	R	GYRO_YOUT_L[7:0]
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT_H[15:8]
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT_L[7:0]

MPU-6500 Register Map and Descriptions -Page 7-

Signal logging and processing

After waking up the sensor and configuring it, raw accelerometer and gyroscope data were read and transmitted over UART2 at a baud rate of 115200. The data stream was monitored using PuTTY, while Python was used to parse and visualize the sensor readings. *(Python plotting scripts are included in the `src/` folder with a simple main code to send data through UART2)*

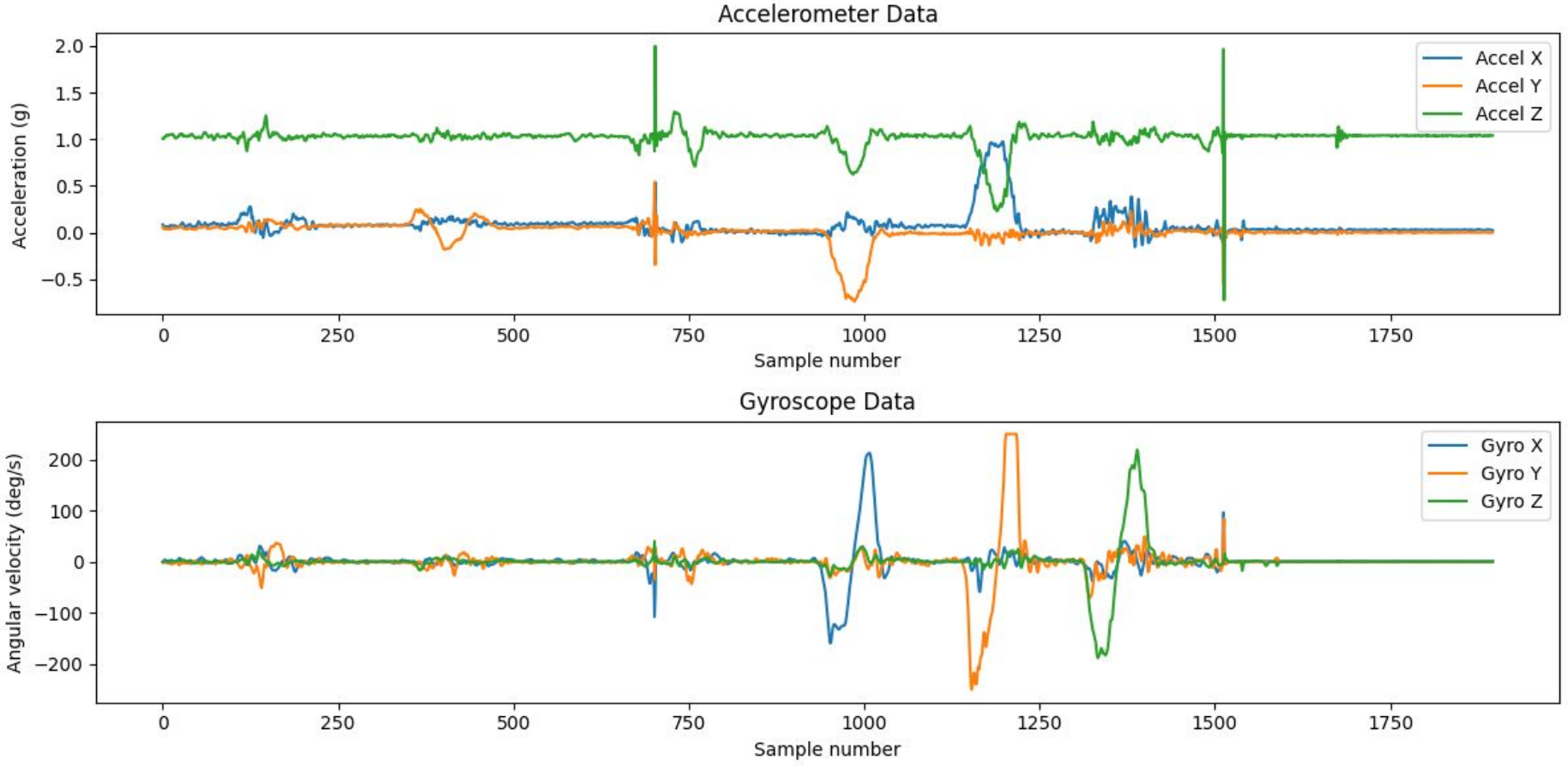
Initial plots showed consistent but noisy signals. As expected from physics, tilting the board in:

- Roll causes changes primarily in the X and Z axes (`ax`, `az`)
- Pitch affects the Y and Z axes (`ay`, `az`)

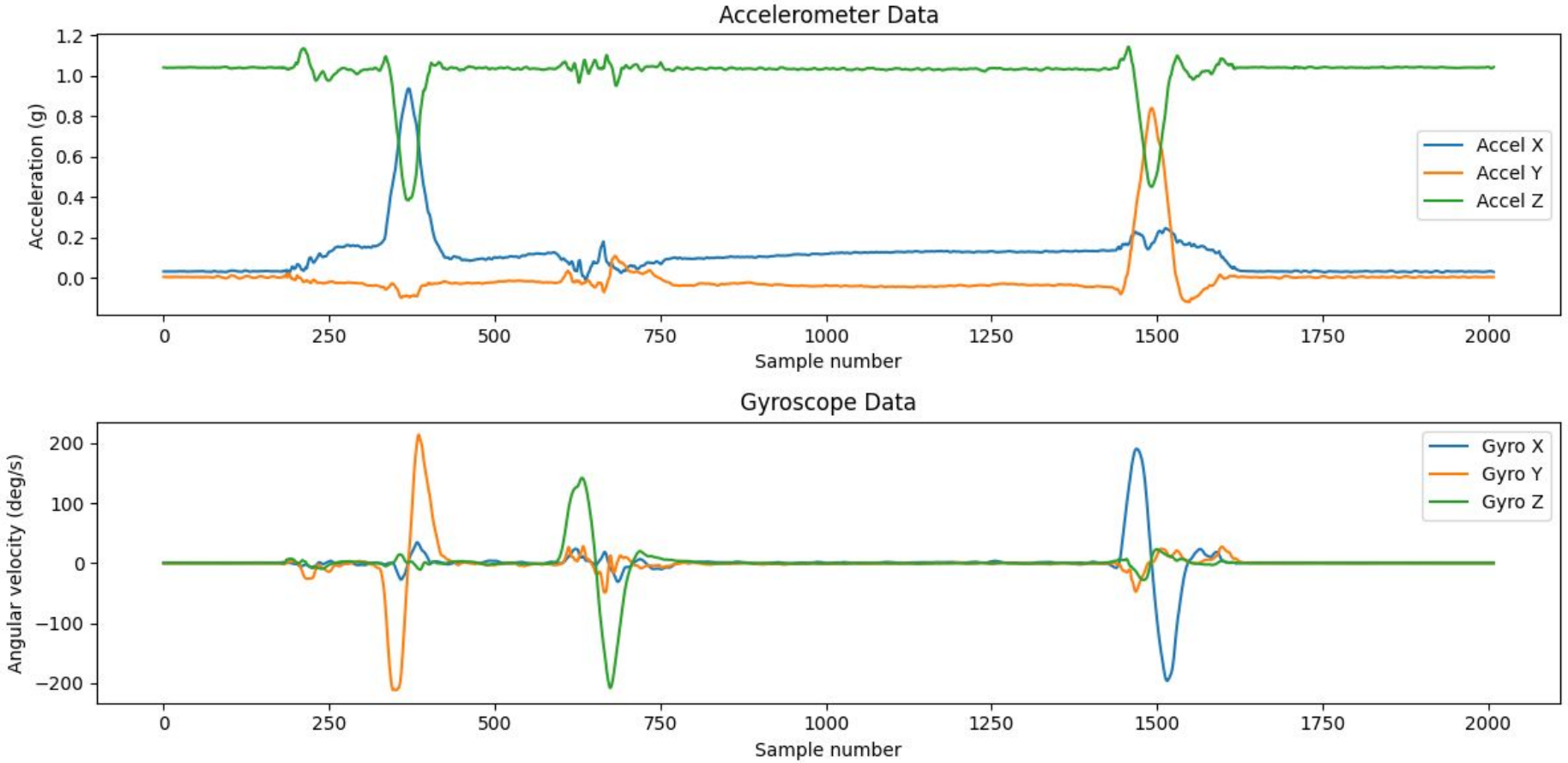
This behavior aligns with how accelerometers sense orientation with respect to the gravity vector.

To reduce the noise and extract usable trends, a simple low-pass filter (LPF) was applied:

$$\text{filtered} = \alpha \cdot \text{new sample} + (1 - \alpha) \cdot \text{previous filtered}$$



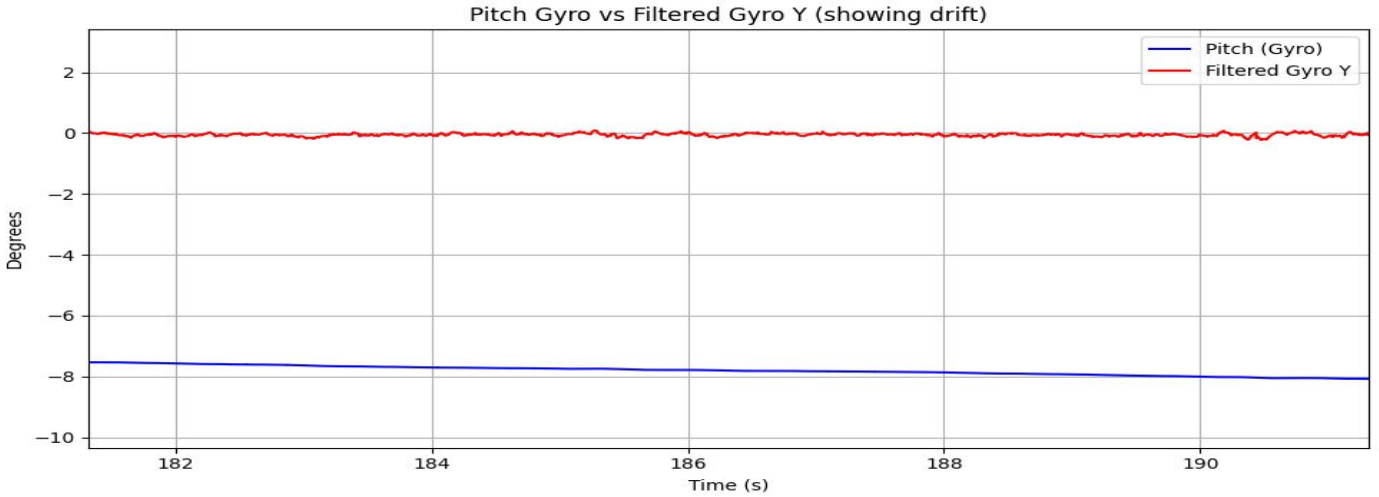
Raw Data (noisy)



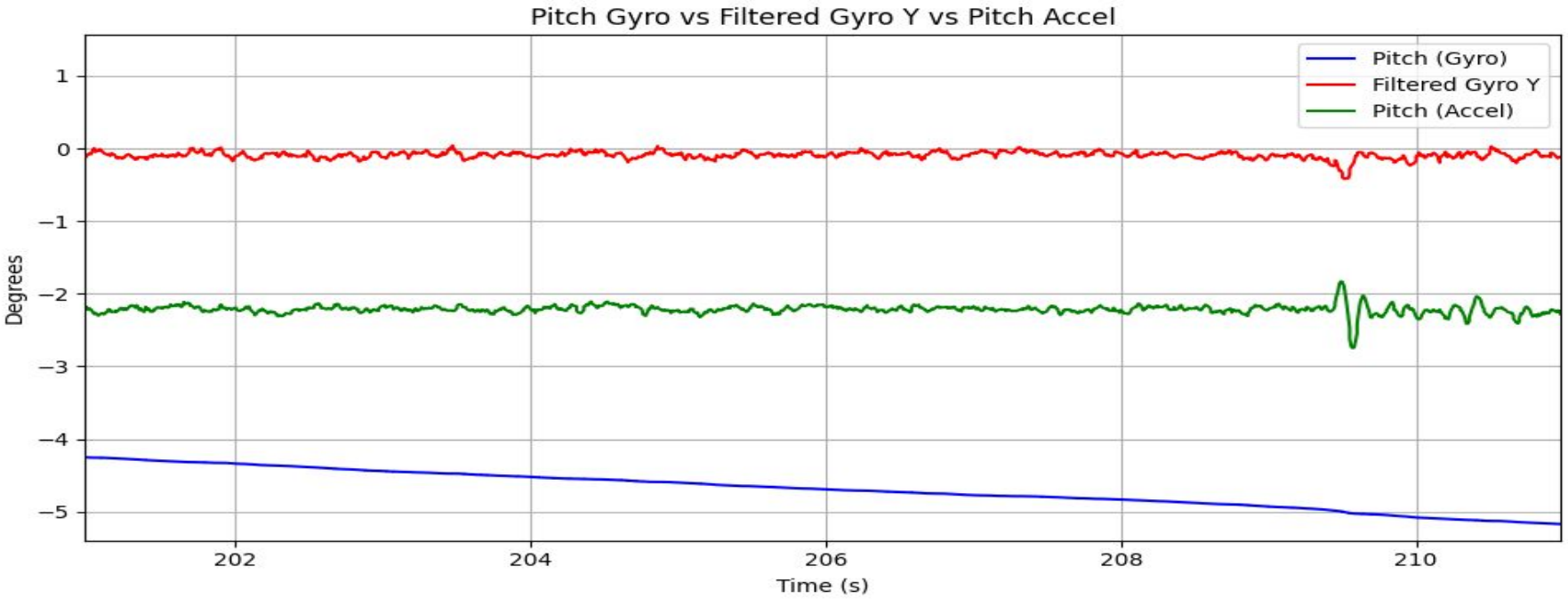
Filtered Data

This might seem fine at first, but once we try to extract roll and pitch angles from the raw data, several issues arise:

- To estimate angles from the accelerometer, we apply trigonometric functions like arctangent. This method provides readings that are stable in the long run, since the accelerometer always senses gravity. However, this introduces more noise to our measurements .
- When using the gyroscope, we estimate orientation by integrating angular velocity over time. This produces smooth and responsive readings. However, even small measurement errors accumulate during integration, leading to a drift in the estimated angle over time (long term error)



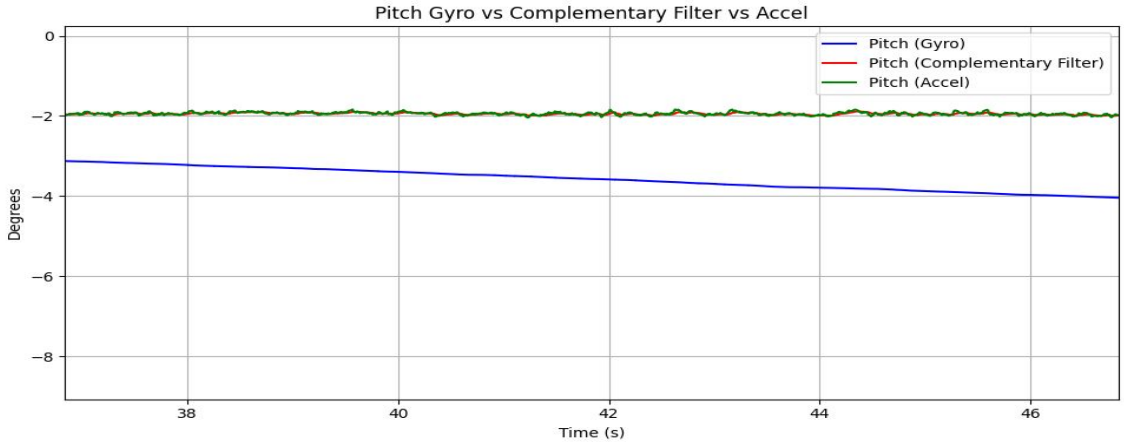
Small errors in gyro reading causing drift while stationary Gyro.ay = 0



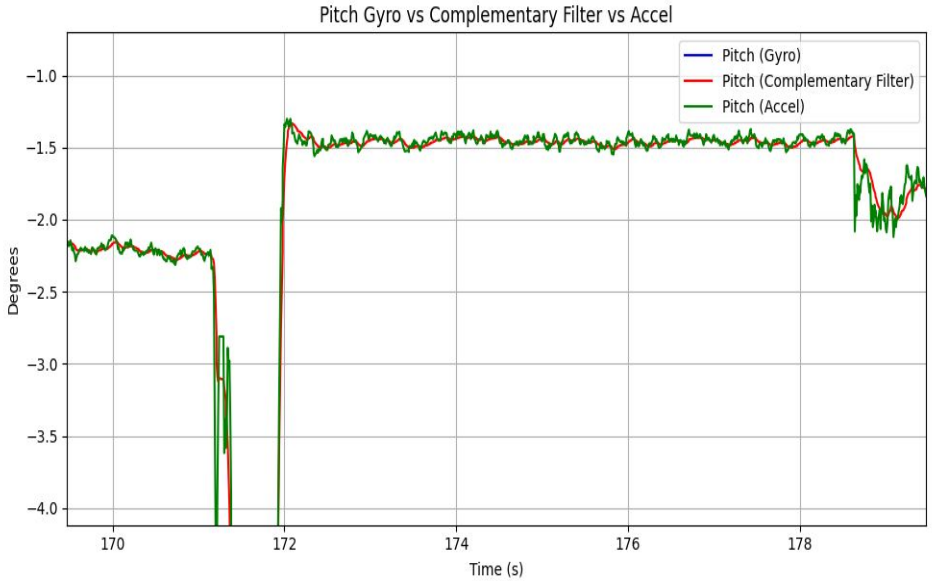
Accelerometer vs gyroscope inferred angles while stationary Gyro.ay = 0

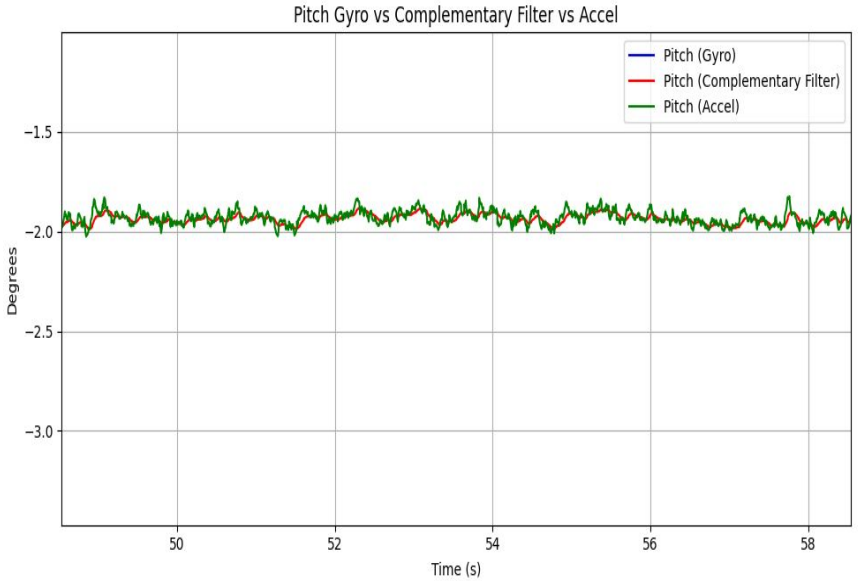
To overcome the limitations of using either the accelerometer or gyroscope alone, a simple and effective solution is to apply a complementary filter. This filter combines the strengths of both sensors by dynamically correcting itself in real-time.The complementary filter is computationally lightweight, making it ideal for embedded systems like the STM32. It performs well in most scenarios involving basic tilt estimation and introduces minimal complexity or constraints.

```
pitch = alphac * (pitch +  filtered_gy* dt) + (1 - alphac) * pitch_acc;
roll  = alphac * (roll  + filtered_gx * dt) + (1 - alphac) * roll_acc;
```



Drift-Free Pitch Estimation via Complementary Filter





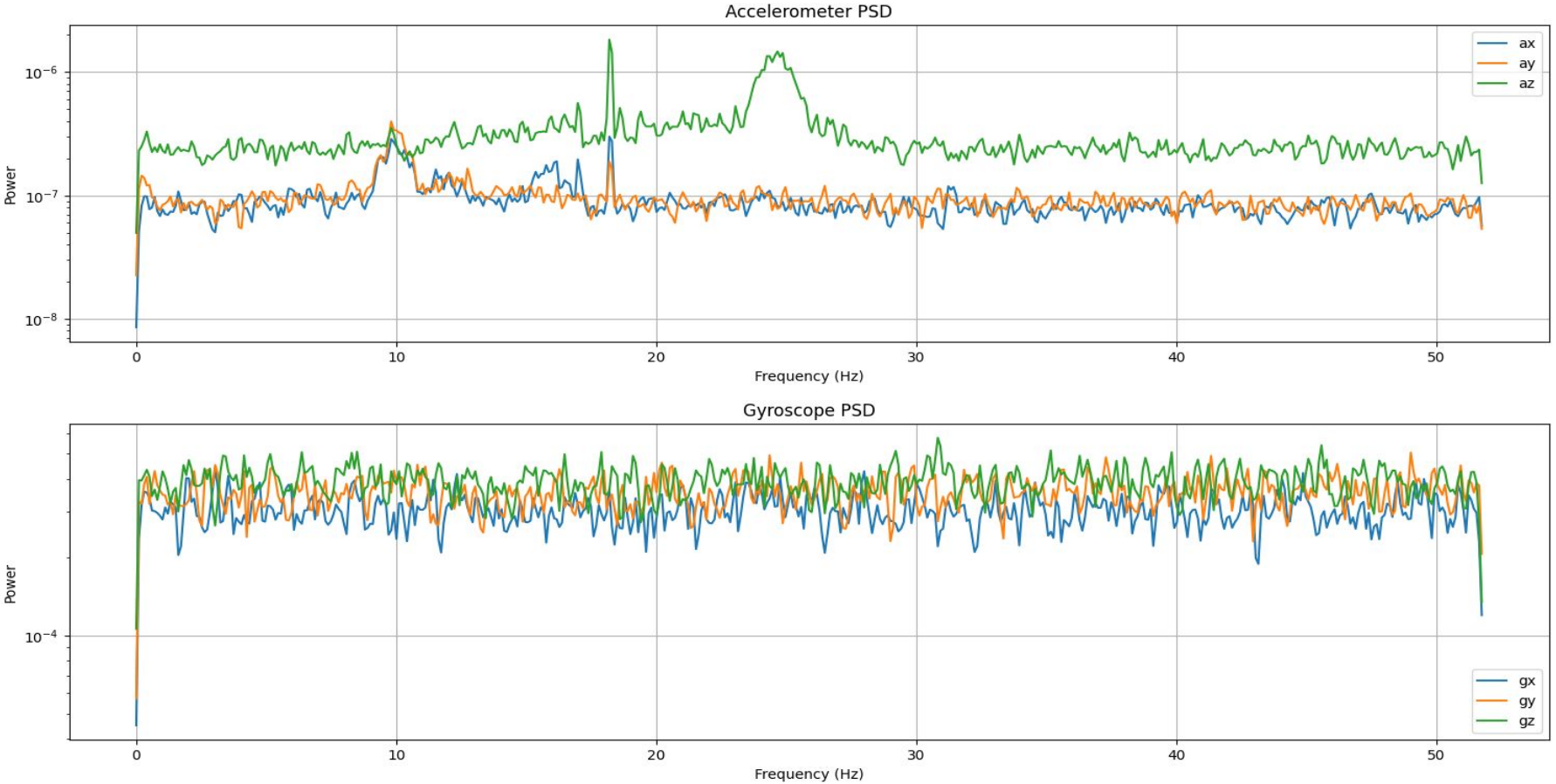
Pitch Angle (Zoomed) – High Stability After Fast Motion

Kalman Filter for Improved Sensor Fusion

While the complementary filter performs well in many scenarios, it has limitations when motion high accuracy is required. It does not adapt to changes in sensor noise, and it treats the weighting between sensors as a fixed constant (`alphac`), which may not be optimal in all conditions.

To address this, a more advanced filtering technique called the Kalman filter was implemented. Unlike the complementary filter, the Kalman filter uses a mathematical model of the system's dynamics along with a statistical model of sensor noise. It continuously updates its internal state and dynamically adjusts how much it trusts each sensor.

To extract statistical characteristics of the sensor noise, we recorded sensor readings while the device was stationary for five minutes. The data logging was performed using a Python script (available in the `src/plotpsd` folder), which captured the sensor outputs and saved them into a CSV file for further analysis. This logged data was then processed to calculate power spectral density.



PSD Plot of Logged Sensor Data

From the PSD plot, we observe that the noise is not white (the PSD is not flat). The accelerometer shows approximately 30% variation with noticeable peaks at certain frequencies, while the gyroscope data is more consistent but still exhibits around 50% variation.

Because the PSD did not provide us with useful statistical features , we instead calculated the variance of the gyroscope raw data and the variance of the accelerometer-inferred angles. This was done under the assumption of ergodicity, allowing us to treat temporal features as statistical features . These variances were then used to design the Kalman filter

```
Mean and Variance of IMU axes:
ax: mean = 5.243143e-02, variance = 4.700463e-06
ay: mean = 3.836754e-03, variance = 4.998939e-06
az: mean = 1.043089e+00, variance = 1.607885e-05
gx: mean = 1.157736e+00, variance = 1.571491e-02
gy: mean = -2.807326e-02, variance = 1.843767e-02
gz: mean = 1.856110e-01, variance = 2.033479e-02
PS C:\Users\nabil> ^C
PS C:\Users\nabil>
```

```
PS C:\Users\nabil> python -u "c:\Users\nabil\OneDrive\Desktop\plot.py"
Accelerometer-based roll: mean = 3.678436e-03 rad, variance = 4.596579e-06 rad^2
Accelerometer-based pitch: mean = -5.022306e-02 rad, variance = 4.306421e-06 rad^2

Measurement noise covariance R matrix:
[[4.59657852e-06 0.00000000e+00]
 [0.00000000e+00 4.30642060e-06]]
PS C:\Users\nabil>
```

Implementation

The process noise covariance matrix Q represents the uncertainty in the system dynamics, which in this case is primarily due to the gyroscope noise and drift. We set Q based on the variance of the gyroscope angular velocity measurements.

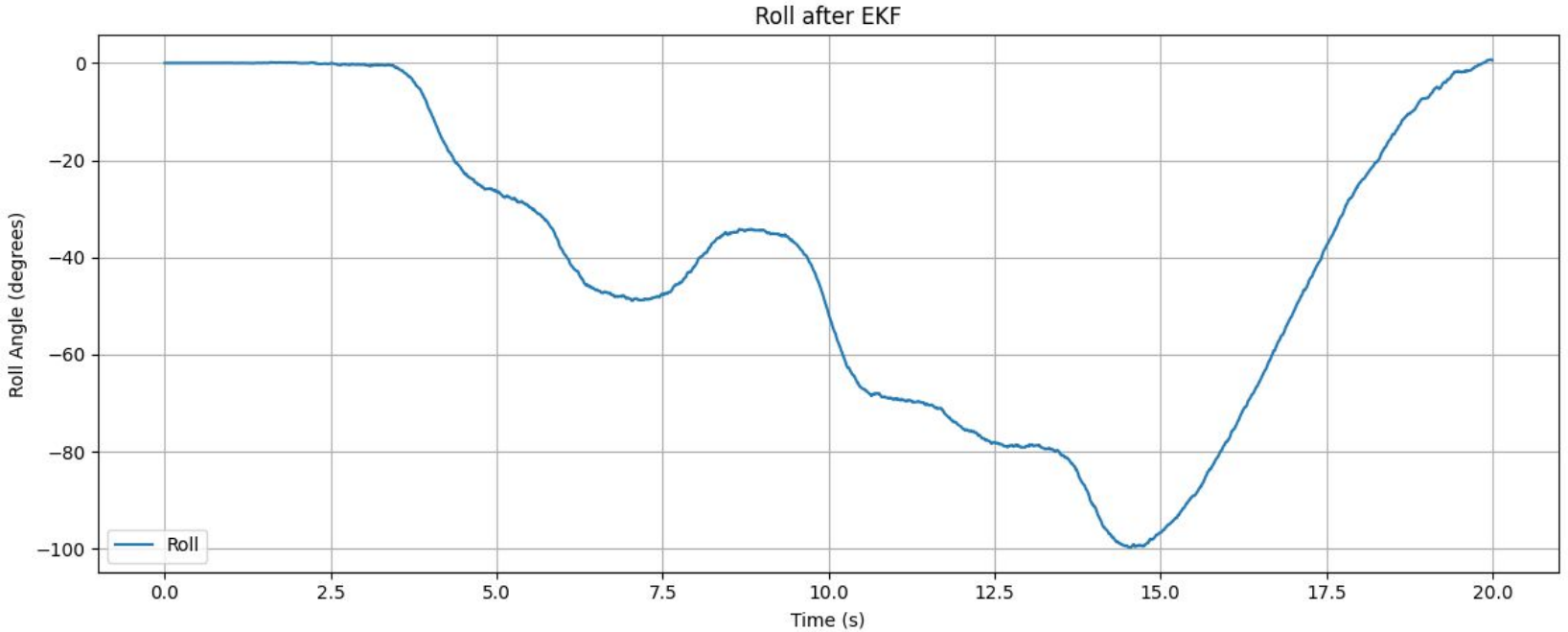
The measurement noise covariance matrix R captures the uncertainty in the measurement, which comes from the accelerometer-derived angle estimates. We set R using the variance of these accelerometer angles

The matrix A is the Jacobian of the nonlinear function T(x) with respect to the state vector x. It captures how small changes in the current roll and pitch angles influence the predicted change in those angles.

The matrix P represents the error covariance of the state estimate. It quantifies the uncertainty associated with the current estimate of the state vector x

The Kalman filter uses these matrices to predict the system state and its uncertainty, then updates the prediction based on sensor measurements, balancing trust between model and data. This allows for accurate, noise-robust estimation of roll and pitch angles from noisy IMU data.

The steps are as follows :



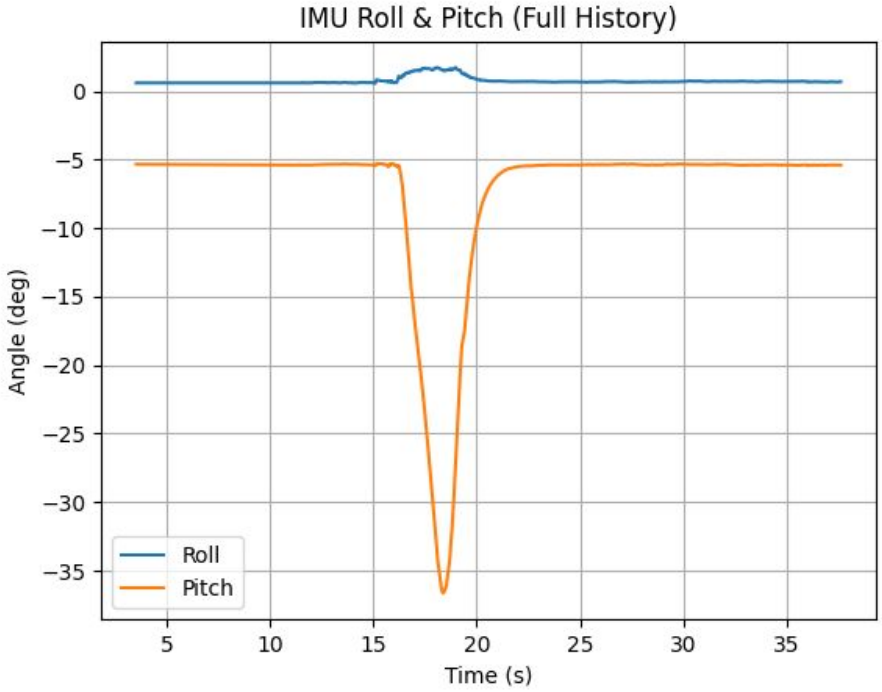
Roll Angle Estimated by EKF: Noise-Free and Responsive Tracking

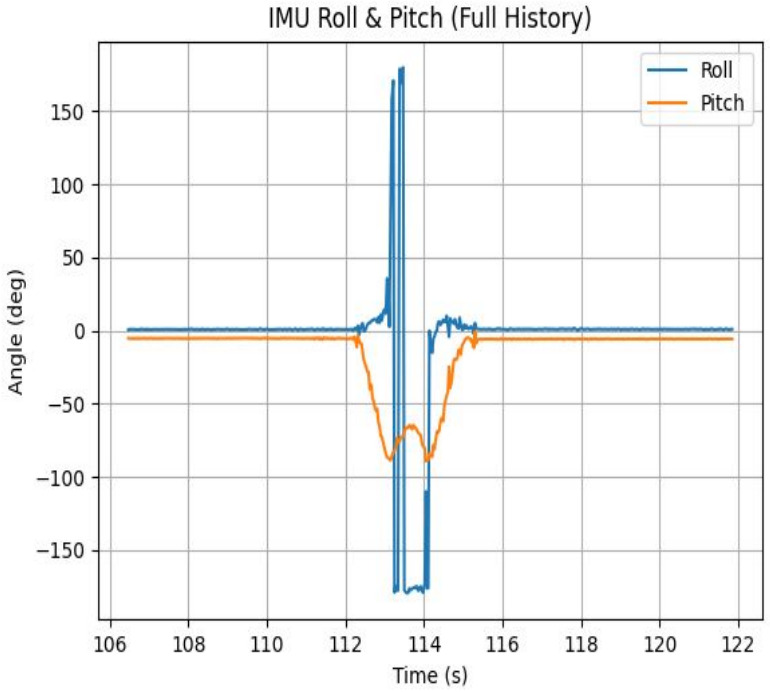
Gimbal Lock

- The first screenshot shows the system at approximately -35° pitch, where the roll angle exhibits only minor variations (mostly because of my hand movement imperfection). This indicates stable and reliable orientation estimation in moderate pitch ranges.

- The second screenshot captures the system near -90° pitch, where gimbal lock occurs. Here, the roll angle estimate becomes unstable and exhibits erratic fluctuations, reflecting the loss of a degree of freedom in the Euler angle representation.

To address this limitation, one approach is to freeze the roll angle estimation at steep pitch angles, making the system suitable for applications involving only moderate rotations. For scenarios involving larger pitch angles, quaternion-based representations can be introduced, as they avoid gimbal lock and provide robust orientation tracking across the full range of motion.





Effect of Gimbal Lock on Roll Angle Estimation at Different Pitch Angles