IS THIS *"THE SPHERE"* ?

NO, IT IS NON-CIRCULAR CUBE PROJECT

THE CUBE

September 6, 2024

Project Repository:
github.com/EfeAydinalp

Efe AYDINALP

1 General Project Overview

This project is focused on interfacing with an MPU6050 gyroscope sensor using an STM32F407G-DISC1 microcontroller to gather, process, and transmit sensor data for real-time visualization in a Python-based 3D cube simulation.

1.1 MPU6050 Sensor Setup:

The MPU6050 sensor is connected to the STM32 microcontroller via the I2C communication protocol. This sensor provides raw data for acceleration and angular velocity (gyroscope data) across the three axes: X, Y, and Z. The sensor is initialized and configured using the I2C protocol to set the full-scale ranges for the gyroscope and accelerometer.

1.2 Data Acquisition and Processing:

The microcontroller reads raw data from the MPU6050 sensor, which is initially in a format that requires conversion to human-readable and usable units. The raw data from the accelerometer and gyroscope are converted into proper units (g for acceleration and degrees per second for angular velocity). To reduce noise and improve the stability of the readings, a simple low-pass filter is applied to the converted data. This filtering process smooths out rapid changes in the sensor data, providing more accurate and stable readings.

1.3 Data Transmission via UART:

After processing the sensor data, the microcontroller transmits the filtered data via the UART (Universal Asynchronous Receiver-Transmitter) protocol to a Python script running on a connected computer. The UART protocol is chosen for its simplicity and effectiveness in serial communication between the microcontroller and the computer.

1.4 Real-time 3D Simulation in Python:

The Python script receives the sensor data and uses it to manipulate a 3D cube simulation. The cube's orientation is updated in real-time based on the gyroscope data from the MPU6050 sensor, allowing the user to visualize the sensor's movements as a rotating cube on the screen. This simulation provides an intuitive representation of the sensor's orientation in space, showing how the MPU6050 reacts to different movements.

2 Detailed Explanation of C Code Functions

2.1 'mpu6050-init()'

- ***Purpose:***

Initializes the MPU6050 sensor by configuring its registers and taking it out of sleep mode.

- ***Explanation:***

The function first checks if the MPU6050 is connected and ready using the HAL-I2C-IsDeviceReady() function. It then configures the gyroscope and accelerometer with appropriate sensitivity settings. Finally, it exits the sensor from sleep mode by writing to the power management register.

```c
void mpu6050_init()
{
    //HAL_StatusTypeDef HAL_I2C_IsDeviceReady(I2C_HandleTypeDef *hi2c,
       uint16_t DevAddress, uint32_t Trials, uint32_t Timeout);
    HAL_StatusTypeDef ret = HAL_I2C_IsDeviceReady(&hi2c1, DEVICE_ADDRESS <<1,
       1, 100);
    if(ret == HAL_OK)
    {
        printf("  MPU6050 is ON!\n");
    }
    else
    {
        printf(" Something went wrong. Check the cables or device please.\n");
    }
        uint8_t temp_data = FS_GYRO_500;
    ret = HAL_I2C_Mem_Write(&hi2c1, DEVICE_ADDRESS<<1, REF_CONFIG_GYRO, 1, &
       temp_data, 1, 100);
    //HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t
       DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData,
       uint16_t Size, uint32_t Timeout);
    //from stm32f4xx_hal_i2c.h file
    if(ret == HAL_OK)
    {
        printf(" 27th register is written correctly and device is configurated
           .\n");
    }
    else
    {
        printf(" Register Problem\n");
    }
    //acceleratior part
    temp_data = FS_ACC_4G;
    ret = HAL_I2C_Mem_Write(&hi2c1, DEVICE_ADDRESS<<1, REF_CONFIG_ACC, 1, &
       temp_data, 1, 100);
```

```c
    //HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout);
    //from stm32f4xx_hal_i2c.h file
    if(ret == HAL_OK)
    {
        printf(" 28th register is written correctly and device is configurated.\n");
    }
    else
    {
        printf(" Register Problem\n");
    }
    //Sleep Mode
    temp_data = 0;
    ret = HAL_I2C_Mem_Write(&hi2c1, DEVICE_ADDRESS<<1, REF_CONFIG_CTRL, 1, &temp_data, 1, 100);
    //HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout);
    //from stm32f4xx_hal_i2c.h file
    if(ret == HAL_OK)
    {
        printf(" Woke up. Ready to humble\n");
    }
    else
    {
        printf(" ZzZzZzZzZzZ...\n");
    }
}
```

2.2 mpu6050-read-raw()

- ***Purpose***:

Reads raw data from the MPU6050 sensor.

- ***Explanation***:

The function reads 16-bit raw data from the accelerometer and gyroscope registers of the MPU6050. This raw data represents the sensor's measurements directly, before any conversion or filtering.

```c
void mpu6050_read_raw()
{
    uint8_t datax[2];
    int16_t raw_accel_x;
```

```
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA, 1, datax, 2, 100);
    raw_accel_x = ((int16_t)datax[0]<<8)+datax[1];
        uint8_t datay[2];
    int16_t raw_accel_y;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA+2, 1, datay, 2, 100);
    raw_accel_y = ((int16_t)datay[0]<<8)+datay[1];
    uint8_t dataz[2];
    int16_t raw_accel_z;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA+4, 1, dataz, 2, 100);
    raw_accel_z = ((int16_t)dataz[0]<<8)+dataz[1];
    int16_t raw_gyro_x;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA_GYR, 1, datax, 2, 100);
    raw_gyro_x = ((int16_t)datax[0]<<8)+datax[1];
    int16_t raw_gyro_y;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA_GYR+2, 1, datay, 2, 100);
    raw_gyro_y = ((int16_t)datay[0]<<8)+datay[1];
    int16_t raw_gyro_z;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA_GYR+4, 1, dataz, 2, 100);
    raw_gyro_z = ((int16_t)dataz[0]<<8)+dataz[1];
    printf("raw_accel_x =%d  ",raw_accel_x);
    printf("raw_accel_y =%d  ",raw_accel_y);
    printf("raw_accel_z =%d  |  ",raw_accel_z);
    printf("raw_x_gyr =%d  ",raw_gyro_x);
    printf("raw_gyro_y =%d  ",raw_gyro_y);
    printf("raw_gyro_z =%d  \n",raw_gyro_z);

}
```

2.3 mpu6050-read-converted()

- ***Purpose***:

Reads and converts raw data from the MPU6050 sensor into usable units.

- ***Explanation***:

This function reads the same raw data as mpu6050-read-raw() but then converts it into more meaningful units—g's for acceleration and degrees per second for angular velocity—using appropriate scaling factors.

```c
void mpu6050_read_converted()
{
    uint8_t datax[2];
    int16_t raw_accel_x;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA, 1, datax, 2, 100);
    raw_accel_x = ((int16_t)datax[0]<<8)+datax[1];
    uint8_t datay[2];
    int16_t raw_accel_y;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA+2, 1, datay, 2, 100);
    raw_accel_y = ((int16_t)datay[0]<<8)+datay[1];
    uint8_t dataz[2];
    int16_t raw_accel_z;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA+4, 1, dataz, 2, 100);
    raw_accel_z = ((int16_t)dataz[0]<<8)+dataz[1];
    int16_t raw_gyro_x;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA_GYR, 1, datax, 2, 100);
    raw_gyro_x = ((int16_t)datax[0]<<8)+datax[1];
    int16_t raw_gyro_y;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA_GYR+2, 1, datay, 2, 100);
    raw_gyro_y = ((int16_t)datay[0]<<8)+datay[1];
    int16_t raw_gyro_z;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA_GYR+4, 1, dataz, 2, 100);
    raw_gyro_z = ((int16_t)dataz[0]<<8)+dataz[1];
    float converted_accel_x = (float)raw_accel_x / 16384.0;
    float converted_accel_y = (float)raw_accel_y / 16384.0;
    float converted_accel_z = (float)raw_accel_z / 16384.0;
    float converted_gyro_x = (float)raw_gyro_x / 131.0;
    float converted_gyro_y = (float)raw_gyro_y / 131.0;
    float converted_gyro_z = (float)raw_gyro_z / 131.0;
    printf("converted_accel_x =%.2f  ",converted_accel_x);
    printf("converted_accel_y =%.2f  ",converted_accel_y);
    printf("converted_accel_z =%.2f  |  ",converted_accel_z);
    printf("converted_gyro_x =%.2f  ",converted_gyro_x);
    printf("converted_gyro_y =%.2f  ",converted_gyro_y);
    printf("converted_gyro_z =%.2f  \n",converted_gyro_z);
}
```

2.4 mpu6050-read-filtered()

- ***Purpose***:

Reads, converts, and applies a low-pass filter to the MPU6050 sensor data.

- ***Explanation***:

After reading and converting the raw data, this function applies a simple low-pass filter to smooth out the data. The filtered values are then updated for each axis of the accelerometer and gyroscope. This filtering helps in reducing the noise and making the data more stable.

```
void mpu6050_read_filtered()
{
    uint8_t datax[2];
    int16_t raw_accel_x;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA, 1, datax, 2, 100);
    raw_accel_x = ((int16_t)datax[0]<<8)+datax[1];
    uint8_t datay[2];
    int16_t raw_accel_y;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA+2, 1, datay, 2, 100);
    raw_accel_y = ((int16_t)datay[0]<<8)+datay[1];
    uint8_t dataz[2];
    int16_t raw_accel_z;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA+4, 1, dataz, 2, 100);
    raw_accel_z = ((int16_t)dataz[0]<<8)+dataz[1];
    int16_t raw_gyro_x;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA_GYR, 1, datax, 2, 100);
    raw_gyro_x = ((int16_t)datax[0]<<8)+datax[1];
    int16_t raw_gyro_y;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA_GYR+2, 1, datay, 2, 100);
    raw_gyro_y = ((int16_t)datay[0]<<8)+datay[1];
    int16_t raw_gyro_z;
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS<<1)+1, REF_DATA_GYR+4, 1, dataz, 2, 100);
    raw_gyro_z = ((int16_t)dataz[0]<<8)+dataz[1];
    float converted_accel_x = (float)raw_accel_x / 16384.0;
    float converted_accel_y = (float)raw_accel_y / 16384.0;
    float converted_accel_z = (float)raw_accel_z / 16384.0;
    float converted_gyro_x = (float)raw_gyro_x / 131.0;
    float converted_gyro_y = (float)raw_gyro_y / 131.0;
    float converted_gyro_z = (float)raw_gyro_z / 131.0;
    accel_x_filtered = ALPHA * converted_accel_x + (1.0 - ALPHA) * previous_accel_x;
```

```
    accel_y_filtered = ALPHA * converted_accel_y + (1.0 - ALPHA) * previous_accel_y;
    accel_z_filtered = ALPHA * converted_accel_z + (1.0 - ALPHA) * previous_accel_z;
    gyro_x_filtered = ALPHA * converted_gyro_x + (1.0 - ALPHA) * previous_gyro_x;
    gyro_y_filtered = ALPHA * converted_gyro_y + (1.0 - ALPHA) * previous_gyro_y;
    gyro_z_filtered = ALPHA * converted_gyro_z + (1.0 - ALPHA) * previous_gyro_z;
    // Update for the previous values with the current ones for the next iteration
    previous_accel_x = accel_x_filtered;
    previous_accel_y = accel_y_filtered;
    previous_accel_z = accel_z_filtered;
    previous_gyro_x = gyro_x_filtered;
    previous_gyro_y = gyro_y_filtered;
    previous_gyro_z = gyro_z_filtered;
    printf("accel_x_filtered =%.2f  ",accel_x_filtered);
    printf("accel_y_filtered =%.2f  ",accel_y_filtered);
    printf("accel_z_filtered =%.2f  |  ",accel_z_filtered);
    printf("gyro_x_filtered =%.2f  ",gyro_x_filtered);
    printf("gyro_y_filtered =%.2f  ",gyro_y_filtered);
    printf("gyro_z_filtered =%.2f  \n",gyro_z_filtered);
}
```

2.5 mpu6050-read-and-send-filtered()

- ***Purpose***:

Reads filtered data from the MPU6050 and sends it via UART.

- ***Explanation***:

This function combines the functionality of reading, converting, filtering the data, and then formatting it into a string. The formatted data string is then transmitted over UART to a connected Python script. This is the key function for the real-time data communication aspect of the project.

```
 mpu6050_read_and_send_filtered()
{
    uint8_t data[14];
    int16_t raw_accel_x, raw_accel_y, raw_accel_z;
    int16_t raw_gyro_x, raw_gyro_y, raw_gyro_z;
    // Read accelerometer and gyroscope data
    HAL_I2C_Mem_Read(&hi2c1, (DEVICE_ADDRESS << 1) + 1, REF_DATA, 1, data, 14, 100);
    // Combine the data bytes into integers
```

```c
    raw_accel_x = (int16_t)(data[0] << 8 | data[1]);
    raw_accel_y = (int16_t)(data[2] << 8 | data[3]);
    raw_accel_z = (int16_t)(data[4] << 8 | data[5]);
    raw_gyro_x = (int16_t)(data[8] << 8 | data[9]);
    raw_gyro_y = (int16_t)(data[10] << 8 | data[11]);
    raw_gyro_z = (int16_t)(data[12] << 8 | data[13]);
    // Convert the raw data to meaningful values
    float converted_accel_x = (float)raw_accel_x / 16384.0;
    float converted_accel_y = (float)raw_accel_y / 16384.0;
    float converted_accel_z = (float)raw_accel_z / 16384.0;
    float converted_gyro_x = (float)raw_gyro_x / 131.0;
    float converted_gyro_y = (float)raw_gyro_y / 131.0;
    float converted_gyro_z = (float)raw_gyro_z / 131.0;
    // Apply filtering (Simple Low-Pass Filter)
    accel_x_filtered = ALPHA * converted_accel_x + (1.0 - ALPHA) * previous_accel_x;
    accel_y_filtered = ALPHA * converted_accel_y + (1.0 - ALPHA) * previous_accel_y;
    accel_z_filtered = ALPHA * converted_accel_z + (1.0 - ALPHA) * previous_accel_z;
    gyro_x_filtered = ALPHA * converted_gyro_x + (1.0 - ALPHA) * previous_gyro_x;
    gyro_y_filtered = ALPHA * converted_gyro_y + (1.0 - ALPHA) * previous_gyro_y;
    gyro_z_filtered = ALPHA * converted_gyro_z + (1.0 - ALPHA) * previous_gyro_z;
    // Update previous values for the next iteration
    previous_accel_x = accel_x_filtered;
    previous_accel_y = accel_y_filtered;
    previous_accel_z = accel_z_filtered;
    previous_gyro_x = gyro_x_filtered;
    previous_gyro_y = gyro_y_filtered;
    previous_gyro_z = gyro_z_filtered;
    // Create a string with the filtered data
    char uart_buffer[100];
    snprintf(uart_buffer, sizeof(uart_buffer), "AX=%.2f AY=%.2f AZ=%.2f GX=%.2f GY=%.2f GZ=%.2f\r\n",
             accel_x_filtered, accel_y_filtered, accel_z_filtered,
             gyro_x_filtered, gyro_y_filtered, gyro_z_filtered);
    // Send the data via UART
    HAL_UART_Transmit(&huart2, (uint8_t*)uart_buffer, strlen(uart_buffer), HAL_MAX_DELAY);
}
```

2.6 MX-USART2-UART-Init()

- ***Purpose***:

Initializes the UART peripheral for communication.

- ***Explanation***:

This function configures the UART peripheral of the STM32 microcontroller. It sets the baud rate, word length, stop bits, parity, and mode (transmit and receive). Proper initialization of UART is essential for reliable data communication between the microcontroller and the Python script.

```
void MX_USART2_UART_Init(void)
{
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart2) != HAL_OK)
    {
        Error_Handler();
    }
}
```

3 Header File(mpu6050.h)

```
#ifndef INC_MPU6050_H_
#define INC_MPU6050_H_

#define DEVICE_ADDRESS 0x68

#define FS_GYRO_250  0
#define FS_GYRO_500  8
#define FS_GYRO_1000 9
#define FS_GYRO_2000 10

#define FS_ACC_2G  0
#define FS_ACC_4G  8
#define FS_ACC_8G  9
#define FS_ACC_16G  10

#define REF_CONFIG_GYRO 27
#define REF_CONFIG_ACC  28
#define REF_CONFIG_CTRL 107
#define REF_DATA        59

#define REF_DATA_GYR        67

#define ALPHA 0.98 // Adjust this value to tune the filter

void mpu6050_init();
void mpu6050_read_raw();
void mpu6050_read_converted();
void mpu6050_read_filtered();
void mpu6050_read_and_send_filtered();

#endif /* INC_MPU6050_H_ */
```

4 IMU-Based Cube Rotation Visualization

4.1 Overview

This Python code creates a 3D simulation of a rotating cube using OpenGL and GLFW, which is controlled by an Inertial Measurement Unit (IMU) connected via a serial port. The cube's rotation is based on the gyroscope data (pitch, roll, and yaw) received from the IMU. The program also includes mechanisms to reduce the effect of sensor noise and drift.

4.2 Code Breakdown

4.2.1 *Importing Required Libraries*

```
import serial
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import glfw
import numpy as np
```

- **serial**: For communicating with the IMU via the serial port.
- **OpenGL.GL, OpenGL.GLUT, OpenGL.GLU**: These libraries provide the necessary functions to render 3D graphics.
- **glfw**: This library is used to create a window and handle input events.
- **numpy**: Although not actively used in this version of the code, it's a powerful library for numerical computations.

4.2.2 *Global Variables*

```
accumulated_pitch = 0.0
accumulated_roll = 0.0
accumulated_yaw = 0.0

THRESHOLD = 4
#1 Radian equals to 57,2958 degree our simulation has 6x const. at first so
    1/6 x 57,2958 equals to 9.5493
#If you want to speed up the simulations's rotation speed you can change the
    constant RADYAN_TO_DEG
RADYAN_TO_DEG = 9.5493
```

- **accumulated-pitch, accumulated-roll, accumulated-yaw**: These variables store the cumulative rotation angles for the cube.

- **THRESHOLD**: A small value used to ignore minor noise and drift from the sensor data. If a detected movement is smaller than this threshold, it will be ignored.

- **RADYAN-TO-DEG**: If this variable's value equals to 9.5493 it means , when you rotate the sensor 90 degree to any direction. The Cube will rotate 90 degree(same degree with your sensor rotation) in same direction. To make simulation more sensitive you can devalue until RADYAN-TO-DEG variable's value equal or higher than 1;

4.2.3 *Parsing IMU Data*

```python
def parse_sensor_data(line):
    try:
        data_parts = line.split()
        data_dict = {}
        for part in data_parts:
            key, value = part.split('=')
            data_dict[key] = float(value)

        pitch = data_dict.get('GX', 0.0)
        roll = data_dict.get('GY', 0.0)
        yaw = data_dict.get('GZ', 0.0)

        return pitch, roll, yaw
    except Exception as e:
        print(f"Error parsing line: {line} -> {e}")
        return None
```

- **parse-sensor-data(line)**: This function reads the raw IMU data line, splits it into individual components, and converts these into pitch, roll, and yaw values based on the corresponding keys ('GX', 'GY', 'GZ'). If there is any error during parsing, it catches the exception and returns None.

4.2.4 *Drawing the Cube*

```python
def draw_cube():
    glBegin(GL_QUADS)
    ...
    glEnd()
```

draw-cube(): This function uses OpenGL commands to draw a colored cube. Each face of the cube is drawn with a specific color using the glColor3f function.

4.2.5 *Applying Threshold to Sensor Data*

```
def apply_threshold(value):
    if abs(value) < THRESHOLD:
        return 0.0
    return value
```

- **apply-threshold(value):** This function ensures that small changes in sensor data (below the threshold) are ignored, preventing the cube from moving due to minor noise or drift. If the absolute value of the input is smaller than the threshold, the function returns 0.0; otherwise, it returns the original value.

4.2.6 *Initializing Pygame for Font Rendering*

```
pygame.init()
pygame.font.init()
font = pygame.font.SysFont("Arial", 18)
```

- **Purpose:** This block initializes Pygame and sets up the font system to render text in the OpenGL window.

- **Explanation:**
  pygame.init(): Initializes all Pygame modules.
  pygame.font.init(): Initializes the font module, allowing the program to render text.
  font = pygame.font.SysFont("Arial", 18): Loads the "Arial" font with a size of 18. This font is used to render text (pitch, roll, yaw) on the screen.

4.2.7 *Rendering Text Using render-text()*

```
def render_text(x, y, text):
    text_surface = font.render(text, True, (255, 255, 255), (0, 0, 0))
    text_data = pygame.image.tostring(text_surface, "RGBA", True)
    glWindowPos2f(x, y)
    glDrawPixels(text_surface.get_width(), text_surface.get_height(), GL_RGBA,
        GL_UNSIGNED_BYTE, text_data)
```

- **Purpose:** Renders text on the screen using Pygame and OpenGL.

- **Explanation:**
  font.render(text, True, (255, 255, 255), (0, 0, 0)): Renders the text in white with a black background.

pygame.image.tostring(...): Converts the rendered text surface to a format that can be used with OpenGL.
glWindowPos2f(x, y): Sets the window position for the text.
glDrawPixels(...): Draws the text as pixels in the OpenGL context.

4.2.8 *Updating the Scene*

```python
def update_scene(pitch_delta, roll_delta, yaw_delta):
    global accumulated_pitch, accumulated_roll, accumulated_yaw
    # Apply the threshold to the deltas
    pitch_delta = apply_threshold(pitch_delta)
    roll_delta = apply_threshold(roll_delta)
    yaw_delta = apply_threshold(yaw_delta)
    # Update the accumulated angles
    accumulated_pitch += (pitch_delta/RADYAN_TO_DEG)
    accumulated_roll += (roll_delta/RADYAN_TO_DEG)
    accumulated_yaw += (yaw_delta/RADYAN_TO_DEG)
    #If accumulated values are higher than 360 degree or lower than -360 degree we set them as zero to reduce some wrong calculations in simulation.
    if(accumulated_yaw > 360 or accumulated_yaw < -360):
        accumulated_yaw = 0
    if(accumulated_roll > 360 or accumulated_roll < -360):
        accumulated_roll = 0
    if(accumulated_pitch > 360 or accumulated_pitch < -360):
        accumulated_pitch = 0
    print(f"Updating scene with Pitch: {pitch_delta}, Roll: {roll_delta}, Yaw: {yaw_delta}")
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    # Set up perspective projection for 3D drawing
    glPushMatrix()  # Push for 3D transformations
    glTranslatef(0.0, 0.0, -5)
    glRotatef(accumulated_roll, 1.0, 0.0, 0.0)
    glRotatef(accumulated_pitch, 0.0, 1.0, 0.0)
    glRotatef(accumulated_yaw, 0.0, 0.0, 1.0)
    draw_cube()
    # Restore the projection matrix for 2D drawing
    glPopMatrix()
    #Switch to orthographic projection for 2D text rendering
    glMatrixMode(GL_PROJECTION)
    glPushMatrix()
    glLoadIdentity()
    gluOrtho2D(0, 800, 600, 0)  # Set up orthographic projection with window size
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
```

```python
    # Render text in the top-left corner (screen coordinates)
    render_text(10, 20, f"Pitch: {accumulated_pitch:.2f}")
    render_text(10, 40, f"Roll: {accumulated_roll:.2f}")
    render_text(10, 60, f"Yaw: {accumulated_yaw:.2f}")
    # Restore perspective projection
    glMatrixMode(GL_PROJECTION)
    glPopMatrix()
    glMatrixMode(GL_MODELVIEW)
    # Swap the buffers to show the new frame
    glfw.swap_buffers(window)
```

update-scene(pitch-delta, roll-delta, yaw-delta): *This function handles the logic for updating the cube's orientation based on the latest IMU data:*

- The deltas (pitch, roll, yaw) are first filtered using the apply-threshold function.
- The filtered deltas are then added to the accumulated rotation angles.
- The OpenGL scene is cleared and reset, and the cube is redrawn with the new rotation values applied.
- Set accumulated values which ¡-360 and ¿360 to zero.
- Render accumulated values on IMU screen with using *render-text()*

4.2.9 *Setting Up the Viewport*

```python
def setup_viewport(width, height):
    glViewport(0, 0, width, height)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(45, width / height, 0.1, 50.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
```

- **setup-viewport(width, height):** This function sets up the OpenGL viewport and perspective projection matrix, defining how the scene will be viewed in the window.

4.2.10 *Main Function*

```python
def main():
    global window
    if not glfw.init():
        raise Exception("GLFW cannot be initialized")

    width, height = 800, 600
    window = glfw.create_window(width, height, "IMU Visualization", None, None)
    if not window:
        glfw.terminate()
        raise Exception("GLFW window cannot be created")

    glfw.make_context_current(window)
    glEnable(GL_DEPTH_TEST)
    setup_viewport(width, height)

    while not glfw.window_should_close(window):
        if ser.in_waiting > 0:
            line = ser.readline().decode('utf-8').strip()
            print(f"Received data: {line}")
            orientation = parse_sensor_data(line)
            if orientation:
                update_scene(*orientation)

        glfw.poll_events()

    glfw.terminate()
```

main(): *The core function of the program, which initializes the GLFW window, sets up the OpenGL context, and enters the main event loop. In each iteration:*

- It checks if new IMU data is available on the serial port.
- If data is available, it reads and parses the data.
- The parsed orientation data is then used to update the scene.
- GLFW's event handling is continuously polled to respond to window events.

4.2.11 *Entry Point*

```python
if __name__ == "__main__":
    ser = serial.Serial(
        port='COM5',         # Replace with your COMX port
        baudrate=115200,
        timeout=1
    )
    main()
```

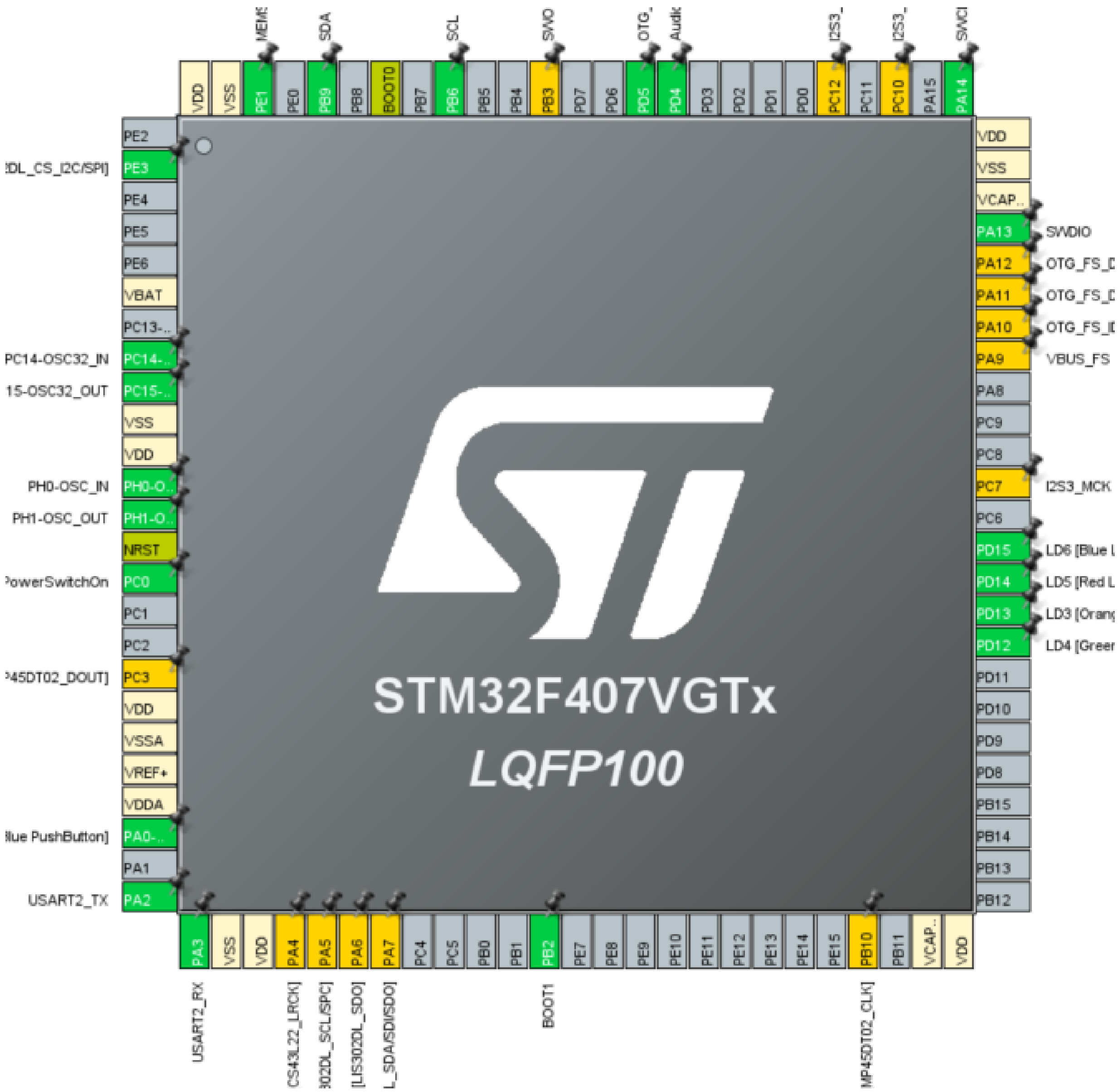
- **Serial Connection Setup:** Before starting the main loop, a serial connection to the IMU is established. The port parameter should be set to the correct COM port where the IMU is connected, and the baud rate must match the IMU's configuration.

- **Program Entry Point:** The script checks if it is the main module being executed, and if so, it establishes the serial connection and starts the main function.

5 Conclusion

The project effectively combines hardware and software to create a real-time, interactive system that visualizes sensor data in a 3D environment. The MPU6050 sensor, connected to an STM32 microcontroller, provides accurate orientation data, which is processed and transmitted via UART to a Python script. The script then updates the orientation of a 3D cube on the screen, providing a clear and intuitive visualization of the sensor's movements. This setup could be used for various applications, such as in robotics, motion tracking, or virtual reality systems.

6 Notes



- *STM32 Pinouts, in this project we only use USART2-TX(PA2), USART2-TX(PA3), SDA(PB9), SCL(PB6)*

7 References

- Sypghere logo
- Datasheet STM32F407G-DISC1
- Errata sheet STM32F407G-DISC1
- Refferance Manual STM32F407G-DISC1
- MPU-6050 Datasheet
- https://freeglut.sourceforge.net
- https://cmake.org
- STM32CubeIDE
- Hercules, it's a serial port terminal and you can use it to check uart com. in this project
- Here is a link so you don't harm yourself in the process