Data Interface - I2C and Gyroscope Sensor

Project Description

The goal of this project was to create a system that reads data from the MPU6050 Gyroscope sensor and communicates with an ESP32 module using I2C communication to establish a master-slave network. Additionally, the ESP32 communicated wirelessly with a local server on the computer using the UDP protocol. The project also included real-time data visualization, where a 3D cube rotated based on the pitch and roll angles calculated from the sensor data. The system built for this project includes an ESP32 connected to the home Wi-Fi network, enabling wireless data transmission to the local server. The MPU6050 provided 3-axis acceleration, temperature, and gyroscope data, with the acceleration data used to compute the pitch and roll angles. A Python-based UDP server on the computer is responsible for receiving and interpreting the sensor data from the ESP32, ensuring efficient communication and real-time feedback.

Issues and Solutions

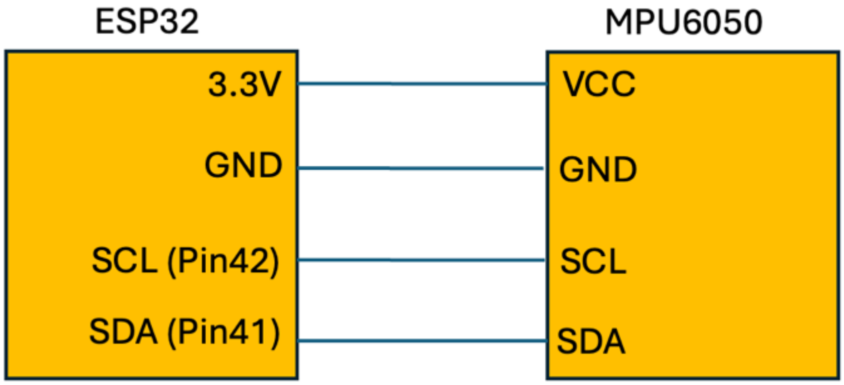
1) **ESP32 board not receiving data from the MPU6050 sensor**: To debug this issue, an I2C connection monitor code was used to scan for I2C-enabled connected devices and print their status to the serial monitor. The code indicated no I2C devices were connected, even though the sensor received power from the ESP32 3.3V and ground pins. Initially, the focus was on debugging the software, verifying the code, and running the sample code provided to read the sensor data. However, the expected data was not received. Hardware issues were suspected. At first, the MPU sensor was replaced, but the same result was observed. The only other option is ESP32-related issues, suspecting that the ESP32's pins weren't correctly soldered. Using another ESP32 board, new pins were soldered onto the board, reconnected to the sensor, and finally received data, which confirmed that poor pin connections on the original board caused the issue.
2) **Incorrect data received on the server:** After establishing communication between the ESP32 and the server to receive sensor data, a difference was observed between the data printed on the serial port and the data received by the server. Initially, it was suspected that the data was being incorrectly packed into the buffer before sending the UDP packet. The buffer values were printed in binary to troubleshoot and confirmed to match on the ESP32 side. The server side, however, showed differing values. Further investigation showed that the issue was related to how data was unpacked on the server. The unpack function from the Python struct library uses a format parameter to determine the return type and byte order. After research, it was discovered that the ESP32 buffer packed the bits with the MSB at the lowest index while the server was interpreting the bits in reverse order. By modifying the unpack function on the server side with the format ">f", the bytes were successfully extracted with the correct MSB placement, resulting in accurate data interpretation.
3) **Create a 3D Cube to Visualize the Pitch and Roll of the Gyroscope:** After receiving accurate data, the next challenge involved translating the pitch and roll angles into meaningful cube rotations and smoothing the transitions between frames. Several strategies were explored to address these challenges. First, if-statements were implemented to stabilize the cube's rotation at low or unstable pitch and roll values, allowing rotation only if the angles were less than -5 or greater than 5. This approach did not yield the desired results. Second, experiments were conducted to adjust the timing

between frame updates, which successfully produced smoother and more stable cube rotations. To visualize the rotation of the cube easier, it was limited to rotating only one direction at a time based on whether the absolute value of the pitch or the roll has a higher value and executing the rotation in that direction.

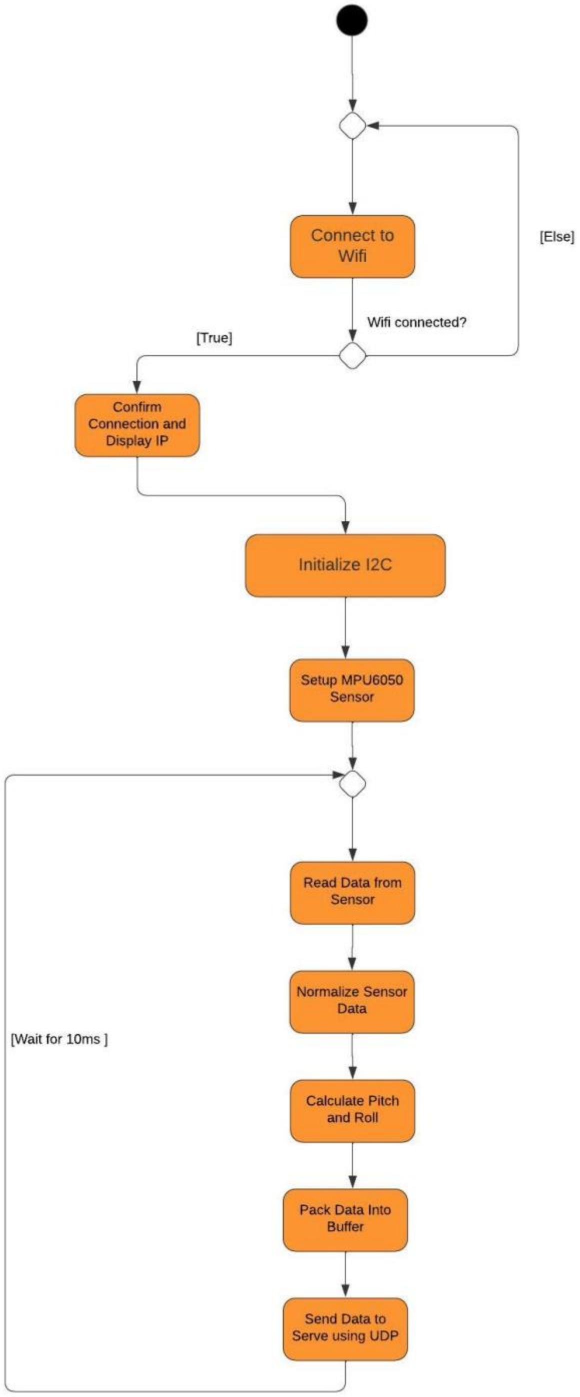
There are many real-world applications for this project across many industries including automotive, robotics, virtual reality, fitness, and aviation. In the fitness industry, this system can be used to provide real-time feedback to users during rehabilitation and physical therapy. By using this system, patients can receive real-time data on their positions to measure balance and maximizing recovery. It can also be used by athletes to measure performance such as speed, agility, and quickness.

In conclusion, this project provided hands-on experience with hardware (ESP32, MPU6050) and software (UDP protocol, data visualization) integration. The outcome of this project included I2C communication was successfully established between the ESP32 module and MPU6050 sensor. A stable UDP server successfully received and interpreted data from the ESP32. The 3D cube responded to the pitch and roll angles, displaying real-time rotation based on sensor data. Through this project, I learned how to create and troubleshoot a local server to communicate to the ESP32 module for wireless communication using Python to implement the UDP protocol. Working with the UDP protocol also deepened my understanding of byte order and data packing/unpacking in network communication. I also gained practical experience using I2C communication to create and debug a master-slave communication network to read data from a sensor.

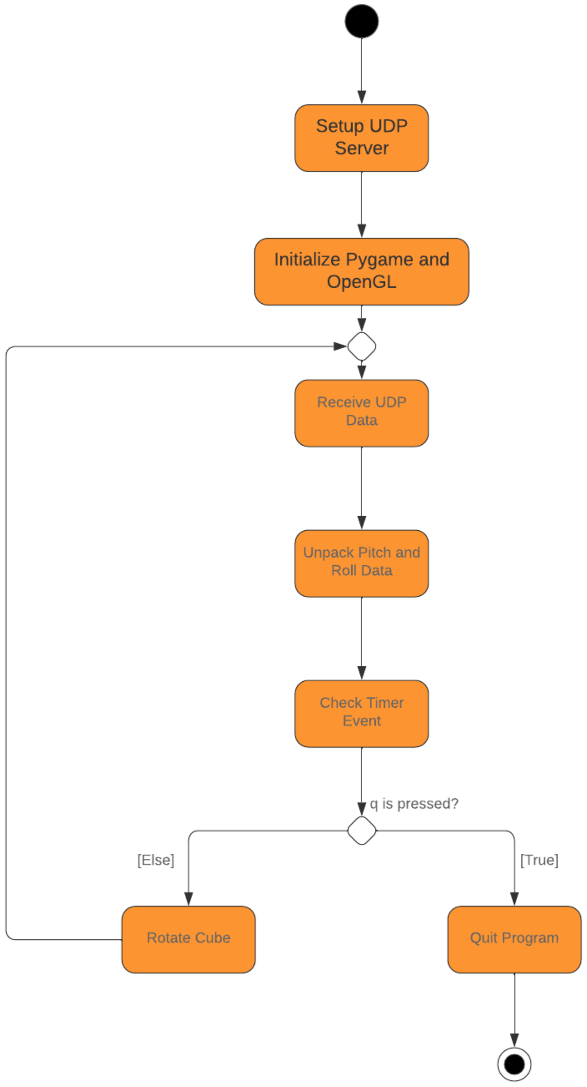
<u>Diagrams</u>



ESP32 to MPU6050 pin connection



Project_I2C.ino – ESP32 and MPU6050 Flow Chart



App1.py – Server and Cube Control Flow Chart