Escola Superior de Tecnologia e Gestão

ESP32-based motorcycle motion tracking with accelerator and gyroscope

Pedro Souza - 26699 | Darya Martsinouskaya - 26610

Master's program in Informatics Engineering and Internet of Things (IoT)

Course Unit: Cyber-phisical systems

Teacher: Dr. João Carlos Martins

Beja. 2025

CONTENTS

FIGURES

INTRODUCTION

Motorcycle accidents are significantly more frequent and severe compared to those involving other vehicles due to the rider's exposure and the lack of protective barriers. Monitoring motorcycle motion can provide valuable insights into rider behavior, motorcycle stability, and potential safety risks. The rise of IoT-based systems enables efficient data collection and analysis, allowing for improved safety and performance evaluation.

This project focuses on developing an ESP32-based motorcycle motion monitoring system that utilizes an accelerometer and gyroscope to collect motion data. The system transfers this data to a local server, enabling later analysis of rider behavior rather than real-time feedback. The system integrates key components that work together to improve rider safety and motorcycle performance:

- Motion sensors (for example, MPU6050 module) capture real-time data on acceleration, angular velocity, and tilt angles.
- ESP32 microcontroller processes sensor data and transmits it via WiFi.
- Data processing module analyzes and is able to store data on a local server, remote system or cloud.

The objective is to gather insights into acceleration patterns, tilting angles, and overall riding dynamics, which can be useful for understanding riding habits, identifying unsafe conditions, and optimizing riding techniques.

The report is organized as follows. Section 1 provides an overview of existing motorcycle motion monitoring systems, related IoT solutions, and the use of motion sensors in rider behavior analysis. It discusses similar research, highlighting gaps that our project aims to address. In Section 2 we define the specific challenges associated with motorcycle motion monitoring, such as data collection accuracy, sensor placement, and limitations of existing systems. This section also justifies the need for a system that records motion data for later analysis rather than real-time feedback. Section 3 presents the overall design of the system, explaining the hardware and software components used. Section 4 covers the implementation of both hardware and software components. Details on how the system was tested and deployment setup for practical usage are described in Section 5.

1. State of the Art

The motorcycle motion monitoring system integrates several key components that work together to improve rider safety and motorcycle performance. First, we have motion sensors, which capture real-time data on speed, acceleration, tilt angles, and trajectory using accelerometers, gyroscopes, and GPS modules. The data collected are then processed through data processing and communication modules, often using IoT platforms, to analyze the motorcycle's dynamics and detect unsafe conditions. In addition, the system features user interaction and feedback mechanisms, such as mobile applications or instant messaging platforms, to provide real-time updates and alerts to the rider, ensuring an optimal and safe riding experience. These integrated technologies offer a comprehensive approach to monitoring motorcycle motion and improving safety through intelligent systems.

Motion sensors, including accelerometers, gyroscopes, and GPS modules, are fundamental components in IoT systems to monitor motorcycle motion. These sensors are designed to collect real-time data about various aspects of motorcycle performance, such as speed, acceleration, tilt angles, and trajectory [1]. By analyzing these parameters, IoT systems can provide valuable insights into the rider's behavior and the motorcycle's dynamics, helping to detect unsafe conditions and suggest corrective actions.

Accelerometers are commonly used to measure the acceleration forces on a motorcycle, both linear and angular. They help in detecting rapid changes in speed or sudden jerks that may indicate aggressive riding, such as sudden accelerations or decelerations. Integration of gyroscopes allows for the measurement of orientation and tilt, helping to track whether the rider is maintaining a safe riding posture or losing control during high-speed maneuvers. GPS modules complement these sensors by providing data on the motorcycle's location and speed relative to the road network. This data is crucial for understanding the context of the riding environment, such as whether the rider is speeding or taking risky routes. By evaluating factors like speed and tilt, they can offer real-time feedback to the rider about optimal speeds and safe navigation strategies, which helps mitigate risk during riding.

For instance, Mr. Rahman J. and Mr. Manoj (2024) in their research explored IoT-enabled crash detection systems that use gyro and GPS sensors to detect accidents and notify emergency services immediately, which can significantly reduce fatalities [2]. Their contribution highlights the importance of real-time response in life-threatening situations. Similarly, Mr. Micko et al. (2021) reviewed sensor systems that track both road conditions and vehicle performance, offering feedback on optimal speed and navigation, which helps mitigate risks during riding [3]. Their work underscores the importance of integrating sensor data to improve safety through better situational awareness and proactive risk mitigation.

However, the field can be further advanced by expanding the scope of sensor integration to include environmental sensors, such as those measuring road conditions and weather. By incorporating these additional data streams, IoT systems could enhance their predictive capabilities, enabling more precise risk assessments in diverse riding conditions. The inclusion of these sensors would not only provide a more comprehensive safety solution but also allow for the identification of potential hazards before they pose a significant threat to the rider.

IoT-based motorcycle motion monitoring systems rely on robust data processing and communication modules to handle sensor data, ensure timely analysis, and provide actionable insights. These components integrate hardware and software to achieve efficient, reliable, and scalable solutions for rider safety and performance optimization. There are three main models of data processing described by researchers: edge computing, cloud computing, and hybrid.

Edge computing focuses on processing data locally, which reduces the delay in transmitting information to centralized systems. Key contributions include the use of edge computing for real-time processing, as demonstrated by Mr. Daraghmi (2022) and Mr. Akhtar et al. (2024). In his research, Mr. Daraghmi (2022) highlighted a vehicle speed monitoring system that utilizes edge computing for immediate data analysis, ensuring swift responses to dynamic traffic conditions [4]. Similarly, Mr. Akhtar et al. (2024) demonstrated the use of lightweight AI models such as SSD Mobilenet for identifying motorbike behaviors on microcontrollers. These models prioritize speed and resource efficiency, enabling operations in environments with limited computational power or bandwidth. By embedding intelligence into edge devices, these systems support real-time feedback, which is essential for scenarios such as detecting aggressive driving or providing rider alerts [5].

In contrast to edge computing model, which processes data locally or near the source, cloud computing plays a pivotal role in handling more computationally intensive tasks, such as large-scale data analytics and behavioral predictions. The cloud computing model is a computational paradigm that allows the delivery of computing services such as storage, processing, and applications over the internet. Unlike traditional computing systems, where data processing occurs locally on individual machines, cloud computing centralizes this process on remote servers, often hosted by third-party providers. Cloud platforms can scale resources based on demand, offering virtually unlimited storage and computational power. This model enables users to access software, services, and data from any location with internet connectivity, providing flexibility and convenience [6].

For instance, Mr. Kristiani et al. (2022) demonstrated the integration of cloud platforms with Narrowband IoT technology for sport cruiser motorcycles. This setup allows data aggregation from multiple sensors, enabling in-depth analytics for monitoring vehicle conditions, rider behavior, and environmental factors. Cloud services also simplify the deployment of predictive models and long-term data storage, which are essential for identifying trends and improving system algorithms over time [1].

Moreover, applications where data from multiple motorcycles need to be synchronized and analyzed collectively, cloud platforms provide unparalleled scalability. For example, Mukhopadhyay et al. (2024) discussed how cloud systems enhance smart transportation by enabling collaborative data processing for multiple vehicles, which contributes to traffic flow optimization and accident prevention [7].

The hybrid model offers a balanced solution, leveraging the advantages of both edge and cloud computing. It could provide an optimal architecture for real-time applications, especially in scenarios where both localized decision-making and centralized analytics are needed. Future research could investigate how to dynamically allocate tasks between edge and cloud resources to optimize energy consumption, cost, and performance. Moreover, improving the coordination between these systems would ensure a more seamless integration for large-scale deployments [8].

Various research studies have explored different user interaction and feedback mechanisms to enhance rider safety and data accessibility in motorcycle motion monitoring systems.

Mobile apps integrate sensors, cloud services, and IoT devices for real-time motorcycle monitoring. Khaskheli et al. (2021) developed a system combining an Android app with cloud-based technologies to track vehicle status and location, integrating GPS, GSM modules, and AI for enhanced monitoring [9]. Such applications improve tracking, reduce theft, and aid regulatory compliance.

Instant messaging services like Telegram and WhatsApp provide real-time alerts and emergency notifications. Setiawan et al. (2024) designed a system that sends Telegram alerts for accidents based on motorcycle tilt detection, ensuring accurate emergency notifications with minimal false alarms [10]. These platforms offer a cost-effective alternative to proprietary apps.

Heads-Up displays (HUDs) in smart helmets project critical data onto a visor, reducing distractions. Mohideen et al. (2022) designed a system incorporating night vision, blind-spot monitoring, and IoT-based accident detection. In case of a crash, it sends alerts via GSM and GPS for rapid emergency response [11]. HUD technology enhances safety and rider awareness through IoT and computer vision innovations.

The state-of-the-art review of existing motorcycle motion monitoring systems highlights the critical role of sensor integration, data processing models, and feedback mechanisms in improving rider safety. However, many solutions prioritize real-time feedback, requiring constant connectivity, which may not be suitable for all users. The proposed ESP32-based system addresses this gap by focusing on local data collection and storage for later analysis. This approach enables comprehensive evaluations of riding behavior without reliance on continuous internet access, offering a practical solution for riders interested in post-ride performance assessments. Future work

can explore hybrid models that balance real-time processing with offline analytics to further enhance motorcycle safety and rider awareness.

2. Problem Analysis

Motorcycle riding presents unique challenges due to its high exposure to external factors, such as road conditions, rider behavior, and motorcycle dynamics. Unlike cars, motorcycles offer less structural protection, making riders more vulnerable to accidents, sudden movements, and road hazards. Effective motion tracking can help analyze riding patterns, improve safety, and optimize performance.

While existing motorcycle monitoring solutions use various sensor-based systems, they often rely on expensive, power-hungry, or complex architectures. This research proposes a cost-effective ESP32-based motion tracking system using an accelerometer and gyroscope to collect and analyze data, offering an accessible solution for riders and researchers.

This chapter explores the limitations of current approaches and the need for an ESP32-based solution, highlighting the benefits of motion tracking using accelerometer and gyroscope sensors.

Many existing motorcycle tracking systems rely on high-end embedded computers, cloud-based processing, or proprietary devices, making them cost-prohibitive for individual riders. Advanced tracking systems are typically integrated into high-end motorcycles, leaving budget-conscious riders without effective monitoring solutions.

Motorcycle motion is inherently dynamic, involving:

- Acceleration and braking patterns (to detect aggressive riding or emergency stops).
- Tilt and lean angles (to monitor cornering stability).
- Sudden shocks or impacts (to detect potholes or crashes).

Accelerometers measure linear acceleration, while gyroscopes detect angular velocity, making them ideal for tracking real-time riding dynamics. Without these sensors, it is difficult to accurately analyze rider behavior and vehicle movement.

Some existing solutions use smartphone-based apps or Bluetooth-connected devices for motion tracking, but they present several issues:

- Smartphone sensors lack precision in detecting real-time tilt and acceleration changes.
- Bluetooth-based systems have a limited range and unstable connectivity.
- Cloud-based solutions introduce latency and require an internet connection.

By using an ESP32 microcontroller with built-in Wi-Fi, our system enables real-time motion data collection and local data transfer without relying on external networks, ensuring low-latency and cost-effective tracking.

Motorcycle IoT systems must be:

- Compact and lightweight to integrate seamlessly into the motorcycle structure.
- Low power-consuming to prevent excessive battery drain.

ESP32 is a highly efficient microcontroller with a low power footprint, making it ideal for real-time data acquisition and wireless transmission in a motorcycle environment.

The ESP32-based motion tracking system addresses these challenges by:

1. Providing an affordable and scalable alternative to expensive tracking solutions.
2. Utilizing an accelerometer and gyroscope for detailed motion analysis.
3. Offering real-time data collection and processing without relying on cloud-based services.
4. Using Wi-Fi for efficient data transmission, avoiding the limitations of Bluetooth or mobile networks.
5. Ensuring power efficiency and compact integration, making it suitable for motorcycles.

Thus, existing motorcycle motion monitoring solutions are either too expensive, reliant on external networks, or lack precise motion tracking. This research introduces a Wi-Fi-enabled ESP32-based system that leverages accelerometers and gyroscopes to track acceleration, tilt, and movement patterns in real time. By providing a cost-effective, standalone solution, this system enhances rider safety, behavioral analysis, and performance monitoring.

The next chapter will describe the system architecture, detailing hardware components, data processing methods, and Wi-Fi communication strategies.

3. System Architecture

This report outlines the characteristics of sensors used alongside an ESP32 microcontroller to monitor movement and tilt. The document explains the system's overall design, the development process for both hardware and software components, and the practical applications of the technology. MEMS (Micro-Electro-Mechanical Systems) sensors, known for their capability to measure acceleration and rotation, are utilized in various fields. In particular, the MPU6050 which integrates a 3-axis accelerometer with a 3-axis gyroscope is employed in applications ranging from motorcycle motion monitoring to stabilization systems in drones, wearable devices, and industrial automation [14,12,15]. These sensors provide essential data on position, speed, and orientation, which supports effective navigation control, stabilization, and detailed movement analysis.

3.1 Hardware

3.1.1 Sensors

Accelerometer (MPU6050).

The accelerometer plays an important role in analyzing motorcycle dynamics by measuring linear acceleration along the X, Y, and Z axes. It detects rapid changes in speed as well as the constant force of gravity, allowing for an accurate assessment of both dynamic movements and static orientation. In practice, the sensor tracks sudden shifts, sharp turns, and vibrations caused by uneven road surfaces [13]. This data proves invaluable for assessing performance and enhancing safety by identifying potentially risky conditions. The accelerometer features adjustable sensitivity settings (ranging from ±2g to ±16g) and provides a high-precision 16-bit resolution on each axis. A sampling frequency of up to 1 kHz enables near real-time data collection, ensuring that rapid changes in motion are accurately recorded.

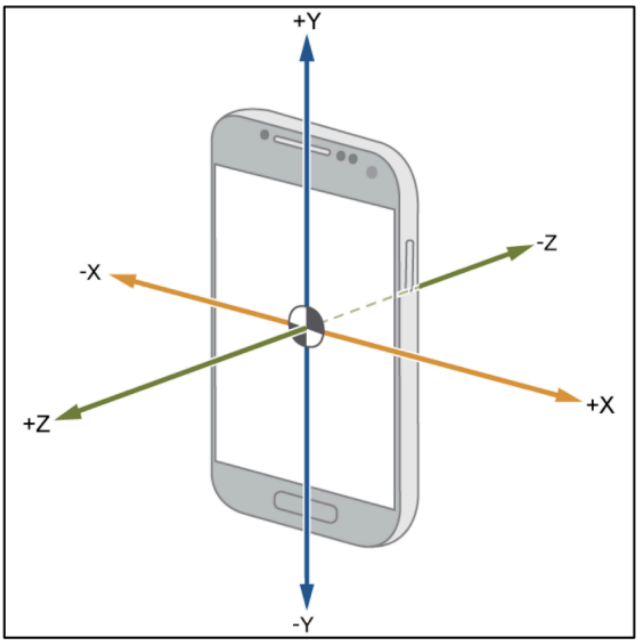

Figure 1. Accelerometer: x, y and z orientation axes

Gyroscope (MPU6050).

Working in tandem with the accelerometer, the gyroscope measures the angular velocity along the X, Y, and Z axes, covering the movements of pitch, roll, and yaw. This measurement is crucial for understanding the motorcycle's tilt, rotation, and overall stability, particularly when negotiating turns or executing dynamic maneuvers. The gyroscope offers an adjustable measurement range (with settings of ±250, ±500, ±1000, and ±2000 °/s) and operates with a

16-bit resolution per axis. In addition, built-in temperature sensitivity helps compensate for any offset variations, ensuring consistent accuracy under different environmental conditions.[15]

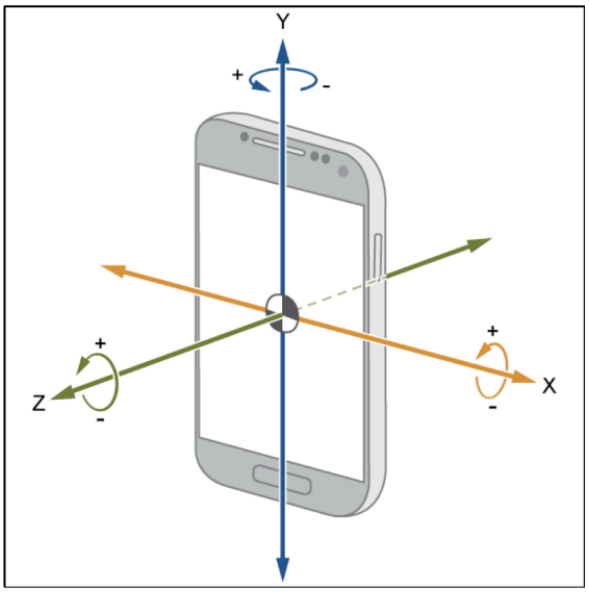

Figure 2. Gyroscope: positive orientation of x, y, and z axes

The MPU6050 sensor is ideal for motorcycle motion monitoring as it provides crucial data for both linear acceleration and rotational motion. The accelerometer component tracks the linear acceleration or deceleration of the motorcycle, providing information on speed changes, bumps, and vibrations. It can also detect orientation changes like tilting, which is valuable for analyzing lean angles during turns.

The gyroscope adds the ability to monitor angular velocity, which tracks the rotational movement of the motorcycle. This makes it essential for detecting tilt, roll, pitch, and yaw movements. In the context of motorcycle motion monitoring, the gyroscope helps track how fast the motorcycle is rotating around its own axis, such as when the rider leans into a curve or adjusts their posture.

Together, the accelerometer and gyroscope in the MPU6050 provide a comprehensive understanding of the motorcycle's dynamics, such as balance, stability, and maneuverability. These sensors are widely used in applications like performance tracking, accident detection, vehicle stability systems, rider safety, and dynamic behavior analysis. The sensor can detect both dynamic accelerations (e.g., sudden accelerations or vibrations from the road) and static accelerations (such as the gravitational force acting on the motorcycle), making it a versatile solution for motorcycle motion analysis [12].

3.1.2 Computational Platform (Remote System/Server/Cloud)

This section outlines the possible solutions for remote systems, servers, or cloud platforms and describes the specific approach used in our project. Possible solutions are:

Possible Solutions includes:

1. Local Server.

A local server hosted on a computer or Raspberry Pi can receive and process data from the ESP32. This approach is cost-effective, provides low latency, and ensures data privacy since no external networks are involved.

Tools: Flask (Python), Node.js, or Apache for server setup; MySQL or SQLite for data storage.

2. Cloud-Based Platforms.

Cloud platforms like AWS IoT, Google Cloud IoT, or Microsoft Azure IoT Hub offer scalable solutions for data storage, processing, and visualization. These platforms provide advanced analytics, machine learning integration, and remote access but may introduce latency and require an internet connection.

Tools: MQTT or HTTP protocols for data transmission; cloud databases like Firebase or AWS DynamoDB.

3. Hybrid Solutions.

A combination of local and cloud-based systems can be used. For example, data can be processed locally for real-time feedback and simultaneously uploaded to the cloud for long-term storage and advanced analysis.

4. IoT Platforms.

Dedicated IoT platforms like ThingSpeak, Blynk, or Ubidots provide pre-built dashboards and analytics tools for visualizing sensor data. These platforms are easy to set up but may have limitations in customization and scalability.

In our project, we use a local server hosted on a computer to receive and process data from the ESP32. The server is accessible via the IP address http://192.168.0.130/, where the motion data is displayed in both numerical and graphical formats. This approach was chosen for the following reasons:

- Low Latency: Data is transmitted and processed locally, ensuring minimal delay.
- Cost-Effectiveness: No additional costs are incurred for cloud services or IoT platforms.

- Privacy and control: Data remains within the local network, ensuring privacy and full control over its usage.

The ESP32 microcontroller collects motion data from the MPU6050 sensor (accelerometer and gyroscope) and transmits it over Wi-Fi to the local server via HTTP POST requests. The ESP32 acts as a web server, hosting a webpage that displays real-time motion data. The data is sent in JSON format, including acceleration (AcX, AcY, AcZ) and angular velocity (GyX, GyY, GyZ) values. The server responds to HTTP requests and provides both numerical data and graphical visualizations using Chart.js. The webpage includes a dynamic chart that updates every second to display trends in acceleration and gyroscope data. The code includes an LED that turns on when specific motion conditions are met (e.g., high acceleration or angular velocity).

3.1.3 Communication Module

ESP32.

The ESP32 microcontroller serves as the central hub for sensor data processing and communication [13]. Data from the MPU6050 is transmitted to the ESP32 via the I2C protocol, which uses just two wires (SDA for data and SCL for clock) to simplify the connection. The built-in Wi-Fi capability (IEEE 802.11 b/g/n) of the ESP32 facilitates the wireless transmission of sensor information to remote servers or cloud services. This feature is especially useful for applications such as motorcycle tracking or accident detection, where immediate access to data can enhance both safety and performance analysis. [14]

Communications Module

The system relies on efficient communication protocols to merge sensor data collection with remote monitoring. The MPU6050 sends information to the ESP32 through I2C, while the microcontroller's Wi-Fi functionality ensures that data can be forwarded to external platforms seamlessly. The software, developed in C++ using the Arduino IDE, makes use of libraries like Wire.h for I2C communication and WiFi.h for managing wireless connectivity. Tools such as Visual Studio Code (VS Code) and the IO platform aid in testing and debugging, ensuring smooth development and integration. [16, 17, 18]

System Integration.

Hardware connections are established with attention to detail: the MPU6050's VCC is linked to the 3.3V output of the ESP32, and its ground (GND) is connected to the ESP32's ground. The sensor's SCL and SDA lines are attached to GPIO22 and GPIO21, respectively, with 4.7 kΩ pull-up resistors ensuring stable I2C communication. A consistent 3.3V power supply guarantees that all components operate within the required voltage parameters.

The combination of high-precision sensor measurements and robust wireless connectivity creates a versatile system capable of detailed motion analysis. The integration of the MPU6050 with the ESP32 offers a reliable platform for real-time monitoring of motorcycle dynamics, supporting advanced applications in safety, performance assessment, and remote data analysis.

3.2 SOFTWARE

In our project, we used the following software components to achieve its functionality.

Arduino IDE was used for writing, compiling, and uploading the firmware to the ESP32 microcontroller. It provides libraries for Wi-Fi communication (WiFi.h), I2C communication (Wire.h), and sensor integration.

ESP32 Firmware is written in C/C++ and runs on the ESP32 microcontroller. It initializes the MPU6050 sensor, reads motion data, and transmits it over Wi-Fi to a local server.

Wi-Fi Communication. The WiFi.h library is used to connect the ESP32 to a local Wi-Fi network. The ESP32 acts as a web server, hosting a webpage that displays real-time motion data.

Local Web Server. The ESP32 hosts a simple web server using the WiFiServer class. The server serves an HTML page with embedded JavaScript for real-time data visualization.

Chart.js. A JavaScript library used to create dynamic, real-time graphs on the webpage. It displays trends in acceleration and gyroscope data over time.

HTML and JavaScript are used to create the user interface for the webpage. JavaScript handles data fetching and updates the graphs and numerical values dynamically.

Serial Monitor is part of the Arduino IDE and used for debugging and monitoring sensor data during development.

- The brief sum up is as follows:
- Development Environment: Arduino IDE.
- Firmware: C/C++ for ESP32.
- Communication: Wi-Fi (WiFi.h), I2C (Wire.h).
- Web Interface: HTML, JavaScript, and Chart.js for real-time data visualization.
- Debugging: Serial Monitor in Arduino IDE.

This software stack provides a lightweight, efficient, and cost-effective solution for real-time motorcycle motion tracking and data visualization.

4. DEVELOPMENT

4.1 HARDWARE

MPU-6050 Pinout includes:

- VCC: Power supply pin, connects to 3.3V (for ESP32, it's recommended to use 3.3V as the ESP32 operates at this voltage level).
- GND: Ground pin, connects to the GND of the microcontroller.
- SDA: Serial Data Line (I2C). This is used for data transmission between the MPU6050 module and the microcontroller.
- SCL: Serial Clock Line (I2C). This pin provides the clock signal to synchronize the data transfer over the I2C bus.
- XDA: Auxiliary data line for I2C communication with external sensors (usually not used in standard projects, can be left unconnected).
- XCL: Auxiliary clock line for external sensors (not typically used, can be left unconnected).
- ADO: I2C Slave Address LSB. This pin is used to change the LSB of the I2C address. By default, the address is 0x68, and if connected to VCC, it changes to 0x69. This pin is not used in your project if you are using the default address 0x68.
- INT: Interrupt pin, used for signaling when data is ready. This pin can be useful for asynchronous data retrieval but can be left unconnected for simple data polling.

The wiring diagram for connecting the MPU6050 to the ESP32 module is as follows (please see also Figure 3 below):

The MPU6050 module communicates with the ESP32 via the I2C interface, requiring only two wires for communication. Connect the SCL and SDA pins of the MPU6050 to D22 and D21 pins on the ESP32. Connect the VCC and GND pins of the MPU6050 to the 3.3V and GND pins of the ESP32 [13].

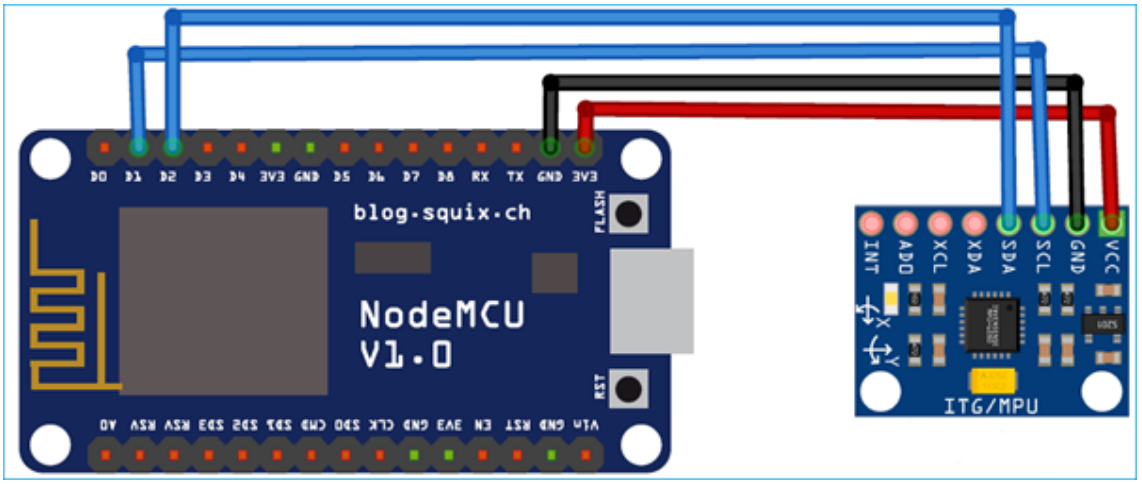

Figure 3. MPU-6050 Pinout [13]

The result is shown on Figure 4 below.

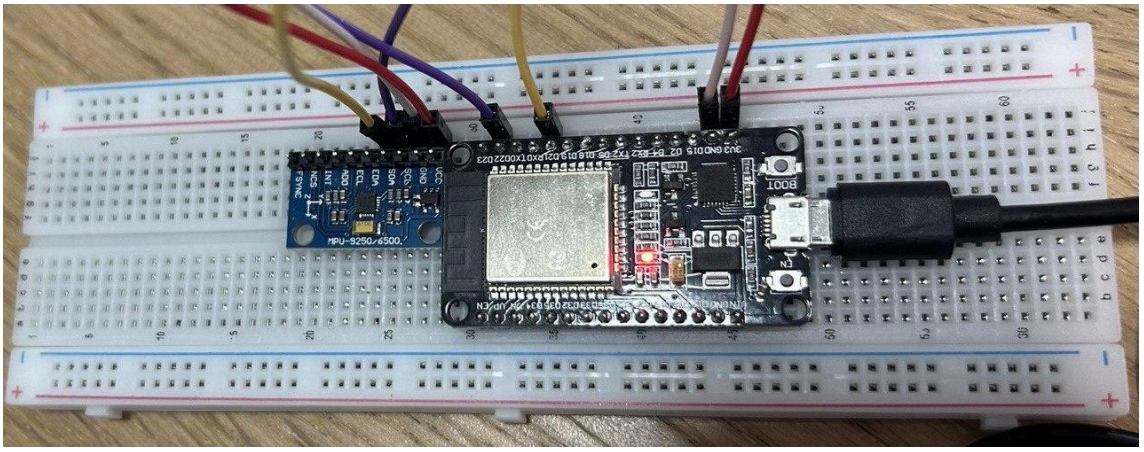

Figure 4. Hardware outcome

4.2 Software

The source code is shown on Figure 5 below.

```
#include <Arduino.h>
#include <Wire.h>
#include <WiFi.h>

// I2C address of MPU-6050
const int MPU_addr = 0x68;
int16_t AcX, AcY, AcZ, GyX, GyY, GyZ;

// Wi-Fi network credentials
const char *ssid = "Alex";
const char *pass = "Sacha3232";
WiFiServer server(80);
```

```
// GPIO for the blue LED
const int ledPin = 2;

// Function declarations
void mpu_read();
void check_conditions();
void handle_client();

void setup() {
Serial.begin(115200);
Wire.begin();
delay(1000);

// Initialize MPU-6050
Wire.beginTransmission(MPU_addr);
Wire.write(0x6B); // PWR_MGMT_1 register
Wire.write(0); // Activate the sensor
Wire.endTransmission(true);
Serial.println("MPU6050 initialized.");

// LED configuration
pinMode(ledPin, OUTPUT);
digitalWrite(ledPin, LOW); // LED off initially

// Wi-Fi connection
Serial.println("Connecting to WiFi...");
WiFi.begin(ssid, pass);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("\nWiFi connected.");
Serial.print("IP Address: ");
Serial.println(WiFi.localIP());

server.begin();
Serial.println("Server started.");
}

void loop() {
mpu_read();
check_conditions();
handle_client();
}

void mpu_read() {
Wire.beginTransmission(MPU_addr);
Wire.write(0x3B); // ACCEL_XOUT_H register
Wire.endTransmission(false);
delay(10);
Wire.requestFrom((uint8_t)MPU_addr, (size_t)14, true); // Specifying types to avoid ambiguity

if (Wire.available() == 14) {
AcX = Wire.read() << 8 | Wire.read();
AcY = Wire.read() << 8 | Wire.read();
```

```
AcZ = Wire.read() << 8 | Wire.read();
GyX = Wire.read() << 8 | Wire.read();
GyY = Wire.read() << 8 | Wire.read();
GyZ = Wire.read() << 8 | Wire.read();
} else {
Serial.println("Error reading MPU6050.");
}

Serial.print("AcX: "); Serial.print(AcX);
Serial.print(", AcY: "); Serial.print(AcY);
Serial.print(", AcZ: "); Serial.print(AcZ);
Serial.print(", GyX: "); Serial.print(GyX);
Serial.print(", GyY: "); Serial.print(GyY);
Serial.print(", GyZ: "); Serial.println(GyZ);
}

void check_conditions() {
float accelX = AcX / 16384.0;
float gyroZ = GyZ / 131.0;

if (fabs(accelX) > 1.0 || fabs(gyroZ) > 100.0) { // Use fabs() for absolute value
digitalWrite(ledPin, HIGH);
Serial.println("LED on: Conditions met.");
} else {
digitalWrite(ledPin, LOW);
Serial.println("LED off: Conditions not met.");
}
}

void handle_client() {
WiFiClient client = server.available();

if (client) {
String request = "";
while (client.available()) {
char c = client.read();
request += c;
}

if (request.indexOf("/data") >= 0) {
client.print("HTTP/1.1 200 OK\r\n");
client.print("Content-Type: application/json\r\n");
client.print("Connection: close\r\n\r\n");
client.print("{");
client.print("\"AcX\":" + String(AcX / 16384.0) + ",");
client.print("\"AcY\":" + String(AcY / 16384.0) + ",");
client.print("\"AcZ\":" + String(AcZ / 16384.0) + ",");
client.print("\"GyX\":" + String(GyX / 131.0) + ",");
client.print("\"GyY\":" + String(GyY / 131.0) + ",");
client.print("\"GyZ\":" + String(GyZ / 131.0));
client.print("}");
} else {
client.print("HTTP/1.1 200 OK\r\n");
client.print("Content-Type: text/html; charset=UTF-8\r\n");
client.print("Connection: close\r\n\r\n");
client.print("<!DOCTYPE HTML><html>");
```

```
client.print("<head>");
client.print("<title>ESP32 Motion Data Monitoring</title>");
client.print("<script src='https://cdn.jsdelivr.net/npm/chart.js'></script>");
client.print("<script>");
client.print("function updateData() {");
client.print("fetch('/data').then(response => response.json()).then(data => {");
client.print("document.getElementById('AcX').innerText = data.AcX;");
client.print("document.getElementById('AcY').innerText = data.AcY;");
client.print("document.getElementById('AcZ').innerText = data.AcZ;");
client.print("document.getElementById('GyX').innerText = data.GyX;");
client.print("document.getElementById('GyY').innerText = data.GyY;");
client.print("document.getElementById('GyZ').innerText = data.GyZ;");
client.print("});");
client.print("}");
client.print("setInterval(updateData, 1000);");
client.print("</script>");
client.print("</head><body>");
client.print("<h1>ESP32 Motion Data</h1>");
client.print("<p><strong>Acceleration on X-axis (forward/backward):</strong> <span id='AcX'>0</span>
g</p>");
client.print("<p><strong>Acceleration on Y-axis (left/right):</strong> <span id='AcY'>0</span> g</p>");
client.print("<p><strong>Acceleration on Z-axis (up/down):</strong> <span id='AcZ'>0</span> g</p>");
client.print("<p><strong>Gyroscope X-axis (tilt forward/backward):</strong> <span id='GyX'>0</span>
°/s</p>");
client.print("<p><strong>Gyroscope Y-axis (tilt left/right):</strong> <span id='GyY'>0</span> °/s</p>");
client.print("<p><strong>Gyroscope Z-axis (turning):</strong> <span id='GyZ'>0</span> °/s</p>");
client.print("<canvas id='myChart' width='400' height='200'></canvas>");
client.print("<script>");
client.print("const ctx = document.getElementById('myChart').getContext('2d');");
client.print("const myChart = new Chart(ctx, {");
client.print("type: 'line',");
client.print("data: {");
client.print("labels: [],");
client.print("datasets: [{");
client.print("label: 'AcX', data: [], borderColor: 'rgb(255, 99, 132)', fill: false},");
client.print("{label: 'AcY', data: [], borderColor: 'rgb(54, 162, 235)', fill: false},");
client.print("{label: 'AcZ', data: [], borderColor: 'rgb(75, 192, 192)', fill: false},");
client.print("{label: 'GyX', data: [], borderColor: 'rgb(153, 102, 255)', fill: false},");
client.print("{label: 'GyY', data: [], borderColor: 'rgb(255, 159, 64)', fill: false},");
client.print("{label: 'GyZ', data: [], borderColor: 'rgb(255, 205, 86)', fill: false}");
client.print("]}");
client.print("});");
client.print("function updateChart() {");
client.print("fetch('/data').then(response => response.json()).then(data => {");
client.print("const labels = myChart.data.labels;");
client.print("const datasets = myChart.data.datasets;");
client.print("if (labels.length > 20) {");
client.print("labels.shift();");
client.print("datasets.forEach((dataset) => dataset.data.shift());");
client.print("}");
client.print("labels.push(new Date().toLocaleTimeString());");
client.print("datasets[0].data.push(data.AcX);");
client.print("datasets[1].data.push(data.AcY);");
client.print("datasets[2].data.push(data.AcZ);");
client.print("datasets[3].data.push(data.GyX);");
client.print("datasets[4].data.push(data.GyY);");
client.print("datasets[5].data.push(data.GyZ);");
```

```
client.print("myChart.update();");
client.print("});");
client.print("}");
client.print("setInterval(updateChart, 1000);");
client.print("</script>");
client.print("</body></html>");
}
client.stop(); // Close the connection
}
}
```

Figure 5. Source code

5. System Testing and Deployment

This section describes the testing methodology, results, and deployment process for the ESP32-based motorcycle motion tracking system. The system was rigorously tested to ensure accurate data collection, reliable Wi-Fi communication, and effective data visualization. The deployment process involved setting up the hardware and verifying its functionality in real-world conditions.

Testing Methodology.

Unit Testing. Each component of the system was tested individually to ensure proper functionality.

- MPU6050 Sensor: Verified that the sensor correctly reads acceleration and gyroscope data by comparing outputs with known values.
- Wi-Fi Communication: Tested the ESP32's ability to connect to the local Wi-Fi network and transmit data to the web server.
- Web Server: Validated that the server correctly serves the HTML page and updates the data in real time.

Integration Testing. The system was tested as a whole to ensure seamless interaction between components.

- Verified that sensor data is correctly transmitted to the web server and displayed on the webpage.
- Tested the system's response to sudden changes in motion (e.g., acceleration, braking, and tilting).

Real-World Testing. The system was deployed and tested.

- Monitored the system's ability to capture and display motion data in real time.
- Evaluated the system's performance in terms of accuracy, latency, and reliability.

Testing Results.

Sensor Accuracy. The MPU6050 sensor provided accurate and consistent readings for acceleration and angular velocity. Please see on Figure 6 example sensor data.

```
LED on: Conditions met.
AcX: -3416, AcY: -3416, AcZ: 14700, GyX: -256, GyY: 2539, GyZ: 16807
LED on: Conditions met.
AcX: -5100, AcY: -3860, AcZ: 15392, GyX: -208, GyY: 2839, GyZ: 13383
LED on: Conditions met.
AcX: -5632, AcY: -3364, AcZ: 14164, GyX: -320, GyY: 812, GyZ: 8160
LED off: Conditions not met.
AcX: -6288, AcY: -3396, AcZ: 14312, GyX: -288, GyY: 1799, GyZ: 6900
LED off: Conditions not met.
AcX: -6904, AcY: -3096, AcZ: 14564, GyX: -336, GyY: 1471, GyZ: 3236
LED off: Conditions not met.
```

Figure 6. Example sensor data

Web Server Performance. The web server successfully hosted the webpage and updated the data in real time. The Chart.js graphs dynamically displayed trends in motion data, providing clear insights into rider behaviour as shown on Figure 7 below.

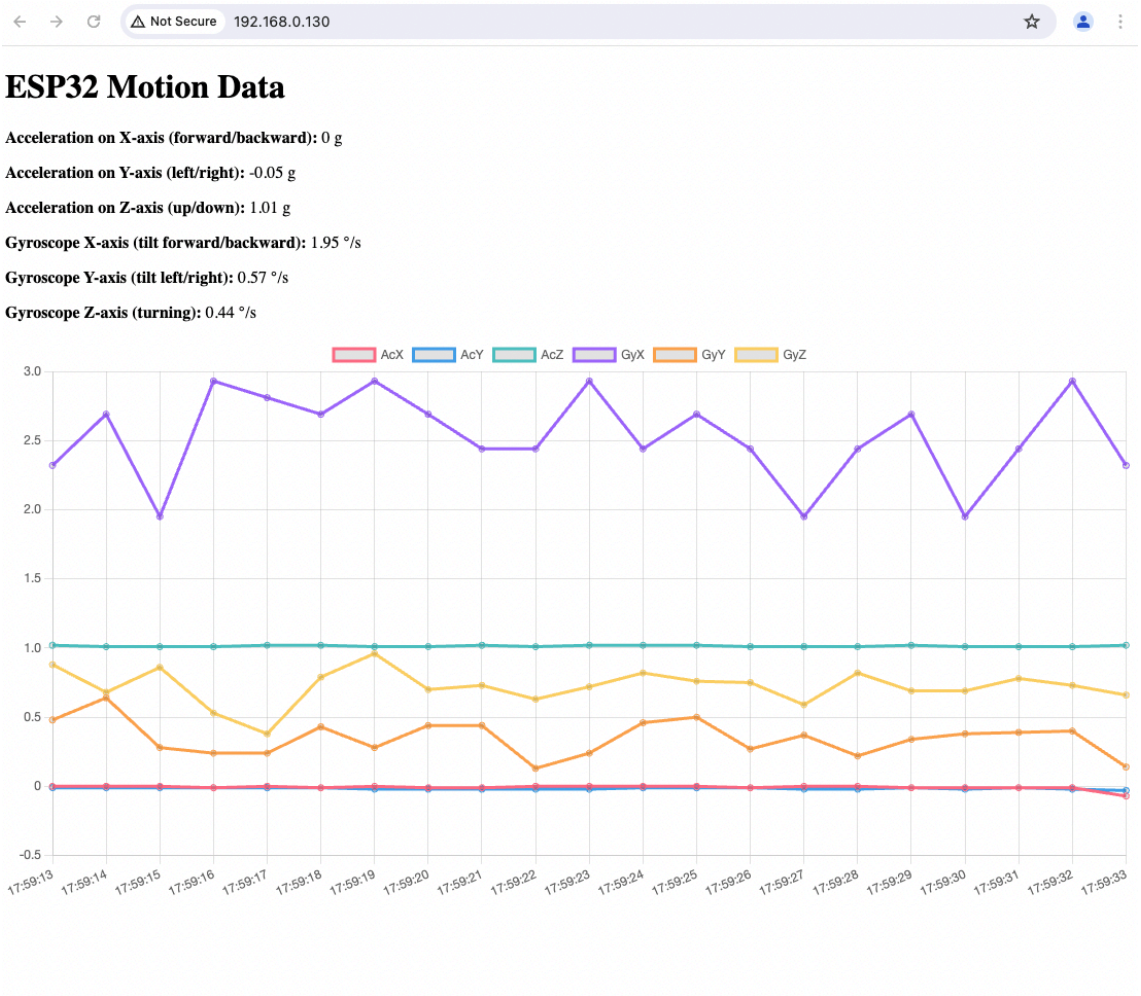

Figure 7. Motion Data webpage output

Real-World Performance. The system is able to perform reliably under various riding conditions, capturing data such as acceleration during braking, tilt angles during cornering, and sudden impacts from road irregularities. The LED indicator is able to detect aggressive riding patterns (e.g., sudden acceleration or sharp turns).

Deployment Process

Hardware Setup:

- The ESP32 and MPU6050 sensor were securely mounted.
- The system was powered using the battery, ensuring continuous operation during rides.

Web Server Access:

- The ESP32's local web server was accessed via the IP address http://192.168.0.130/ from any device connected to the same Wi-Fi network.
- The webpage provided real-time motion data and graphical visualizations, as shown in the screenshot below.

User Training:

- Riders were briefed on how to access and interpret the data on the webpage.
- The system's LED indicator was explained as a simple feedback mechanism for detecting aggressive riding.

Thus, the system testing and deployment phase confirmed that the ESP32-based motorcycle motion tracking system meets its design objectives. The system accurately captures and displays motion data, providing valuable insights into rider behavior and motorcycle dynamics. The successful deployment demonstrates its practicality and potential for real-world applications. Future work could focus on enhancing data storage, integrating additional sensors, and improving the user interface for broader adoption. Additionally, the collected motion data could help identify road hazards like potholes and uneven pavement, enhancing rider safety and aiding infrastructure planning without requiring hardware modifications.

CONCLUSIONS

The ESP32-based motorcycle motion tracking system successfully addresses the challenges of monitoring rider behavior and motorcycle dynamics in a cost-effective and efficient manner. By leveraging the MPU6050 sensor for accurate motion data collection and the ESP32 microcontroller for real-time data processing and Wi-Fi communication, the system provides a robust solution for analyzing acceleration, tilt angles, and angular velocity. The integration of a local web server with dynamic data visualization using Chart.js enables users to monitor and interpret motion data in real time, offering valuable insights into riding patterns and potential safety risks.

Key achievements include:

1. Accurate motion tracking. The system captures precise acceleration and gyroscope data, enabling detailed analysis of rider behavior and motorcycle dynamics.

2. Real-time data visualization. The web server provides a user-friendly interface with real-time graphs and numerical data, making it easy to interpret motion trends.

3. Cost-effectiveness. By utilizing affordable components and avoiding reliance on cloud services, the system offers an accessible solution for individual riders and researchers.

4. Scalability. The modular design allows for future enhancements, such as integrating additional sensors (e.g., GPS) or extending functionality for advanced analytics.

The system has practical applications in:

- Rider safety by identifying aggressive riding patterns and providing feedback to improve safety.
- Performance optimization by analyzing riding techniques to enhance motorcycle performance.
- Research and development by serving as a tool for studying motorcycle dynamics and rider behavior.

Our project demonstrates the potential of IoT-based systems in improving motorcycle safety and performance. By providing an affordable, scalable, and user-friendly solution, the ESP32-based motion tracking system empowers riders and researchers to gain deeper insights into motorcycle dynamics. The successful implementation and testing of the system underscore its practicality and readiness for real-world deployment. Future advancements will further enhance its capabilities, making it an indispensable tool for motorcycle enthusiasts and safety advocates alike.

Bibliographic References

1. T-H Yu, E. Kristiani and Ch-T Yang. «On Construction of Real-Time Monitoring System for Sport Cruiser Motorcycles Using NB-IoT and Multi-Sensors». In: Sensors (2024). URL: https://doi.org/10.3390/s24237484.
2. R. Rahman J. and K. Manoj. «Motorcycle Crash Detection And Alert System Using Iot». In: International Journal of Creative Research Thoughts (IJCRT) (2024). URL: https://ijcrt.org/papers/IJCRT2401295.pdf.
3. I. Zolotova, K. Micko, P. Papcun. «Review of IoT Sensor Systems Used for Monitoring the Road Infrastructure». In: Sensors (2023). URL: https://doi.org/10.3390/s23094469.
4. Y.-A. Daraghmi. «Vehicle Speed Monitoring System Based on Edge Computing». In: Sensors (2022). URL: https://doi.org/10.1109/ICPET53277.2021.00009.
5. M. H. Yousaf A. Akhtar R. Ahmen e S. A. Velastin. «Real-Time Motorbike Detection: AI on the Edge Perspective». In: Mathematics (2024). URL: https://doi.org/10.3390/math12071103.
6. Sh. Jin, Sh. Liang and Y. Chen. «A Review of Edge Computing Technology and Its Applications in Power Systems». In: Energies (2024). URL: https://doi.org/10.3390/en17133230.
7. S. Mukhopadhyay et al. «A Review and Analysis of IoT Enabled Smart Transportation Using Machine Learning Techniques». In: International Journal of Transport Development and Integration 8.1 (mar. de 2024), pp. 61–77. URL: https://www.iieta.org/journals/ijtdi/paper/10.18280/ijtdi.080106.
8. Vaibhav. «Cloud, Edge or Hybrid IoT Solutions: Which Computing Model is Best Suited for your IoT Application?» In: Automotive IoT Blog (2019). URL: https://www.embitel.com/blog/embedded- blog/cloud- edge- or-hybrid- iot- solutions-which- computing- model- is- best- suited- for-your-iot-application.
9. S. A. Khaskheli et al. «AI Based Motor Vehicles Detection and Tracking System Using Smartphone Application». In: International Journal of Advanced Trends in Computer Science and Engineering 10.3 (2021). URL: https://doi.org/10.30534/ijatcse/2021/1121032021.
10. I. P. E. Setiawan et al. «Implementation of Telegram Notification System for Motorbike Accidents Based on Internet Of Things». In: Jurnal Galaksi (Global Knowledge, Artificial Intelligence, and Information System) (2024). URL: https://doi.org/10.70103/galaksi.v1i1.1.
11. A. Mohideen et al. «IoT-Based Smart Helmet». In: International Research Journal of Innovations in Engineering and Technology (2022). URL: https://irjiet.com/common_src/article_file/1672295500_2a1079e8f5_6_irjiet.pdf.

12. InvenSense Inc. “MPU-6000 and MPU-6050 Product Specification. Revision 3.4”. URL: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf. Available: 29.12.2024.
13. AVR microcontrollers: device and programming, literature, designs, practical training. URL: https://microkontroller.ru/esp32-projects/podklyuchenie-giroskopa-i-akselerometra-mpu6050-k-esp32/. Available: 29.12.2024.
14. H. I. M. Ghazalli, M. I. Hassan, Z. A. Zulkifli, S. N. Johari. “MotoSOS: Accident Detection for Motorcycle Riders Using Motion Sensors”. *Journal of Advanced Research in Computing and Applications* 15.1 (2019), pp. 9–19. URL: https://www.akademiabaru.com/doc/ARCAV15_N1_P9_19.pdf.
15. Makerfabs. “MPU6050 3-axis Acceleration Gyroscope 6DOF Module”. URL: https://www.makerfabs.com/mpu6050-3-axis-acceleration-gyroscope-6dof-module.html.
16. Arduino. “Wire Library”. URL: https://www.arduino.cc/en/Reference/Wire.
17. Espressif. “ESP32 Technical Reference Manual”. URL: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf.
18. SparkFun. “I2C Tutorial”. URL: https://learn.sparkfun.com/tutorials/i2c/all.