LAB REPORT 3

MECHATRONICS SYSTEM INTEGRATION

DR. ZULKIFLI BIN ZAINAL ABIDIN

SECTION 1

GROUP 6

MATRIC NUMBER	NAME
2312369	MUHAMMAD FAHIM BIN AHMAD NORISHAM
2312479	MUHAMMAD HAZIMUDDIN BIN FAIRUZ
2312451	MUHAMMAD ZAMARUL AZIM BIN ZULKHIBRI

DATE OF SUBMISSION

Wednesday, 5th November 2025

WEEK 4:

SERIAL INTERFACING WITH MICROCONTROLLER: SENSORS AND ACTUATORS

Abstract

This experiment focuses on designing and **constructing a motion-activated RFID access system** using an Arduino, MPU6050 sensor, RFID module, and a servo motor. In **Task 1**, the goal was to track hand movement using the MPU6050 sensor and display the motion in real-time at PyCharm software (Python language). In **Task 2**, both the motion sensor and RFID reader were combined to control access using motion detection and RFID authentication. This project helped us understand sensor integration, serial communication between Arduino and Python, and how automation systems can use both motion and RFID verification for security.

Keywords: Arduino, MPU6050, RFID, Servo Motor, Motion Detection, Python, Serial Communication

Table of Contents

WEEK 4: SERIAL INTERFACING WITH MICROCONTROLLER: SENSORS AND ACTUATORS

1. Introduction

Nowadays, automation and security systems are becoming crucial and important things. Things like smart doors, motion detectors, and keyless entry systems all rely on sensors and microcontrollers to work. These devices help control access and detect movement automatically, making systems safer and more convenient.

In this lab, we worked on building a smart access system using two main components: an MPU6050 motion sensor and an RFID reader. The MPU6050 can sense movement and position, while the RFID reader can scan special cards or tags to identify a person. Together, they help create a system that reacts to both motion and identity which is similar to how some modern security systems work. The MPU6050 pin is shown in Figure 1.0.

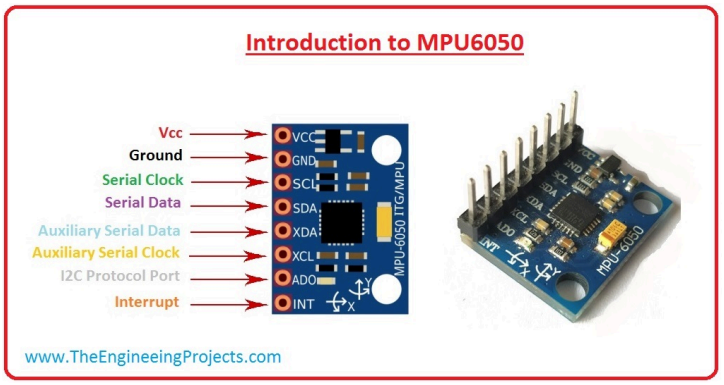


Figure 1.0 Shows the pin for MPU6050

To make everything function, we used Arduino to control the hardware, such as the servo motor (which acted like a door lock) and LEDs (which showed status). On the computer side, we used Python to receive and process live sensor data from the Arduino through serial communication. This allowed the system to send and receive information in real time.

Our main objectives in this experiment were:

- To interface an MPU6050 sensor and RFID module with an Arduino.
- To detect simple motion patterns for MPU6050.
- To integrate motion sensing and RFID authentication to control a servo-based access system.
- To communicate between Python and Arduino over serial.

2. Materials and Equipments

Arduino UNO	1
MPU6050 sensor	1
Light Emitting Diode, LED	2
220 Ω Resistor	1
Servo Motor (SG90)	1
RFID reader (MFRC522) + RFID tags/cards	1
Breadboard	1
Computer with Arduino IDE & PyCharm	1
Jumper Wires	

3. Experimental Setup

3.1 Task 1 - Motion Detection with MPU6050

In this task, we focused on using the MPU6050 motion sensor to detect and track movement. The MPU6050 is a sensor that combines both an accelerometer (which measures acceleration in three directions) and a gyroscope (which measures rotation or tilt). Only 5 pins are used in this task.

Schematic diagram of connection is shown below:

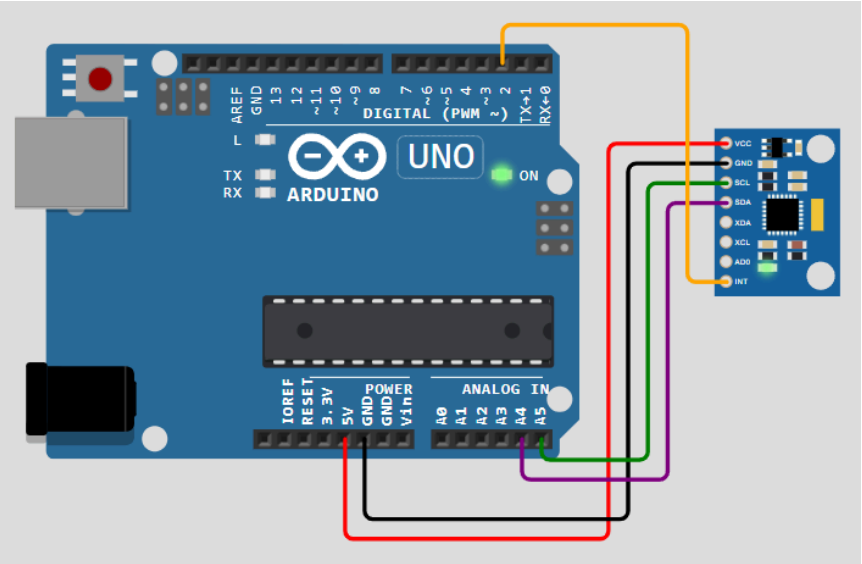


Figure 3.0 Shows the schematic diagram for Task 1 using Wokwi

After connecting the sensor, we uploaded a program to the Arduino that continuously read the sensor values and sent them to the computer via serial communication. On the computer, a PyCharm software was used to receive this data and visualize it using `matplotlib`. The real-time graph showed how the sensor readings changed when the MPU6050 was moved or tilted.

3.2 Task 2

In this task, an RFID reader module was combined with a servo motor, LED indicators, and the MPU6050 sensor to create a simple access control system. The RFID reader was connected to the Arduino through the computer. The servo motor was attached to pin D6, while the green and red LEDs were connected to D5 and D7, each with a current-limiting resistor 220 Ω resistor.

The Arduino was programmed to detect and verify RFID tags. When a valid tag was scanned and the MPU6050 sensor was detecting motion, the green LED turned on, and the servo motor rotated to simulate unlocking a door. If the tag was invalid, the red LED lit up to indicate access denial. The MPU6050 sensor was also connected to the same setup to detect specific motion gestures, adding an extra condition for system activation.

Schematic diagram of connection is shown below:

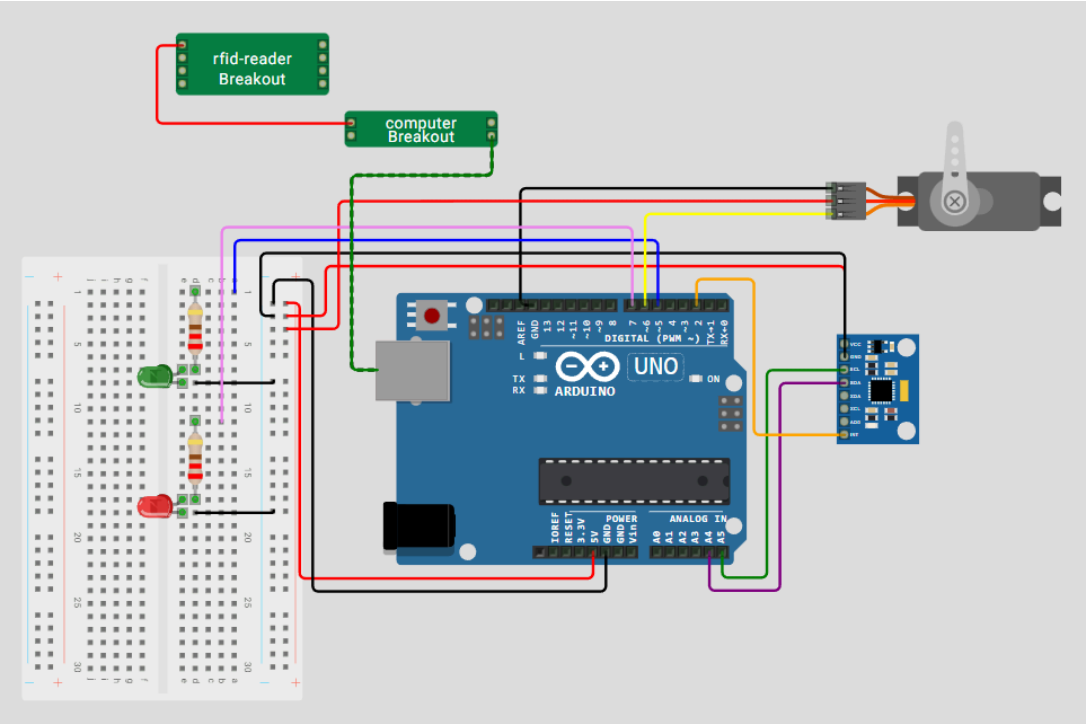


Figure 3.1 Shows the schematic diagram for Task 2 using Wokwi

This setup showed how multiple components could be integrated to perform a coordinated function. The experiment highlighted how the Arduino can act as a central controller that processes input from sensors (RFID and MPU6050) and triggers output devices (servo and LEDs) accordingly.

4. Methodology

4.1 Task 1

4.1.1. Circuit Assembly

All circuit connections were made based on **Figure 3.0**. The MPU6050 motion sensor was connected to the Arduino Uno. The SDA pin of the sensor was connected to A4, and the SCL pin to A5. Power and ground connections were made to the 5V and GND pins of the Arduino, respectively.

The Arduino was powered through the computer's USB port, providing both power and data transfer. Jumper wires were used to make all the necessary connections securely. The setup allowed real-time communication between the MPU6050 sensor and the Arduino for data acquisition.

4.1.2. Serial Communication Setup

The Arduino IDE was used to write and upload the code that collected accelerometer and gyroscope data from the MPU6050. The Arduino continuously read motion data and transmitted it to the computer through the serial port at a baud rate of 9600 bps.

On the computer side, Python (PyCharm) was used to receive the serial data. The Python program utilized the `pyserial` library to establish communication with the Arduino and retrieve live sensor readings. The data was then plotted in real time using `Matplotlib`, allowing observation of the sensor's response to various movements such as shaking or tilting.

4.1.2.1 Code Upload to Arduino IDE

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  mpu.initialize();

  if (!mpu.testConnection()) {
    Serial.println("MPU6050 connection failed!");
    while (1);
  }
}

void loop() {
  int16_t ax, ay, az, gx, gy, gz;
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

  Serial.print(ax); Serial.print(",");
  Serial.print(ay); Serial.print(",");
  Serial.print(az); Serial.print(",");
  Serial.print(gx); Serial.print(",");
  Serial.print(gy); Serial.print(",");
  Serial.println(gz);

  delay(8); // ~125Hz -> faster updates, lower delay
}
```

4.1.3 Python Script

A **Python script** was created in **PyCharm** to receive and visualize the motion data sent by the Arduino. The script used the `pyserial` library to read incoming data and the **Matplotlib** library to generate a real-time graph of motion activity. When the sensor was shaken or moved in a circular motion, the live graph displayed corresponding spikes and oscillations, confirming that the MPU6050 accurately detected motion.

4.1.4. Matplotlib

The Arduino Serial Plotter was used to verify that the MPU6050 was functioning correctly. The live accelerometer and gyroscope readings were displayed as continuously updating waveforms. The baud rate was matched to the Arduino code (9600), and the correct COM port was selected.

When the sensor was moved or rotated, the plotted values changed accordingly, showing variations in acceleration and angular velocity. The Serial Plotter was closed before running the Python script to avoid serial port conflicts.

4.1.4.1 Code Upload to PyCharm

```
      import serial, time, math
      from collections import deque
      import   matplotlib.pyplot   as
plt

# --- Serial connection ---
      ser   =   serial.Serial('COM7',
9600, timeout=1)   # adjust port if
needed

# --- Parameters ---
      GYRO_SCALE = 131.0   # LSB per
°/s for ±250 dps
      DEADZONE = 5.0        # ignore
small gyro noise
      CIRCLE_DEG = 340      # total
rotation threshold for detection
      WINDOW = 2.0          # seconds
of recent data kept
      COOLDOWN = 1.5        # seconds
between detections
```

```
      ))

line_ax.set_ydata(list(data_ax))

line_ay.set_xdata(range(len(data_ay)))

line_ay.set_ydata(list(data_ay))

line_az.set_xdata(range(len(data_az)))

line_az.set_ydata(list(data_az))
              ax.relim()
              ax.autoscale_view()
              fig.canvas.draw()
              fig.canvas.flush_events()

                # --- Gyro integration
---
              now = time.time()
              if last_t is None:
                  last_t = now
```

```
# --- Live plot setup
      plt.ion()
      fig, ax = plt.subplots()
      ax.set_title("MPU6050
Accelerometer Live Data")
      ax.set_xlabel("Samples")
      ax.set_ylabel("Accel (raw)")
      line_ax,   =   ax.plot([],   [],
label='AX')
      line_ay,   =   ax.plot([],   [],
label='AY')
      line_az,   =   ax.plot([],   [],
label='AZ')
      ax.legend()
      data_ax,  data_ay,  data_az  =
deque(maxlen=200),
deque(maxlen=200), deque(maxlen=200)

# --- Gyro integration setup —

      rot_hist = deque()
      # (timestamp, delta_angle)
      last_t,  last_detect  =  None,
0.0

      print("Starting  live  plot  +
circle   detection.\nMove   MPU   in
circle  to  test.  Press  Ctrl+C  to
stop.")

      try:
         while True:
                              raw   =
ser.readline().decode(errors='ignore
').strip()
            if not raw:
                continue

            vals = raw.split(',')
            if len(vals) != 6:
                continue

            try:
                    ax_raw, ay_raw,
az_raw,  gx_raw,  gy_raw,  gz_raw  =
map(int, vals)
            except ValueError:
                continue

# --- Plot accel live ---
            data_ax.append(ax_raw)
            data_ay.append(ay_raw)
            data_az.append(az_raw)

line_ax.set_xdata(range(len(data_ax)
                continue
            dt = now - last_t
            last_t = now

                    gz_dps  =  gz_raw  /
GYRO_SCALE
                    if  abs(gz_dps)  <
DEADZONE:
                gz_dps = 0
            dtheta = gz_dps * dt
      #   incremental   rotation   in
degrees
                rot_hist.append((now,
dtheta))

            # drop old samples
             while rot_hist and now -
rot_hist[0][0] > WINDOW:
                rot_hist.popleft()

            # --- Detection ---
             total = sum(d for _, d in
rot_hist)
               abs_total  =  sum(abs(d)
for _, d in rot_hist)

                    if  (abs(total)  >
CIRCLE_DEG or abs_total > CIRCLE_DEG)
and (now - last_detect > COOLDOWN):
                # direction = "CW" if
total < 0 else "CCW"
                    print(f"✅ Circle
detected!   Direction:   circle,   Net
rotation: {total:.0f}°")
                last_detect = now

      except KeyboardInterrupt:
         print("\nStopped.")
      finally:
         ser.close()
```

4.2 Task 2

4.2.1 Circuit Assembly

The circuit for the integrated RFID and motion access system was built on the same breadboard setup (Figure 3.1). The RFID module was connected to the Arduino Uno.

A servo motor was connected to D6, while two LED indicators were connected to D7 (red) and D5 (green), each with a 220 Ω resistor. The MPU6050 sensor remained connected as in Task 1. The Arduino received power through the computer's USB connection.

4.2.2. Serial Communication Setup

The Arduino handled both the RFID authentication and motion verification processes. When an RFID tag was scanned, its unique ID (UID) was read and sent to the Arduino for verification. If the UID matched a stored value, the Arduino awaited a valid motion input from the MPU6050.

Serial communication was used to send verification data between the Arduino and Python. The Python script monitored the authentication process, displaying whether access was granted or denied based on RFID and motion input conditions.

4.2.3. Code Upload to Arduino IDE

```
#include <Wire.h>
#include <MPU6050.h>
#include <Servo.h>

MPU6050 mpu;
Servo gateServo;

const int greenLedPin = 5;     // authorized LED
const int servoPin = 6;        // servo motor pin
const int redLedPin = 7;       // unauthorized LED

// Motion streaming config
const unsigned long streamDurationMs = 2000;  // 2 seconds
const unsigned long sampleDelayMs = 50;       // ~20 Hz

void setup() {
  pinMode(greenLedPin, OUTPUT);
  pinMode(redLedPin, OUTPUT);
  gateServo.attach(servoPin);
  gateServo.write(90); // start at closed

  Serial.begin(9600);
  Wire.begin();
  mpu.initialize();
  if (!mpu.testConnection()) {
          Serial.println("MPU6050 connection failed!");
    while (1);
  }

   // Default state: red LED ON (idle/unauthorized)
                  Serial.print(ay);
Serial.print(',');
                  Serial.print(az);
Serial.print(',');
                  Serial.print(gx);
Serial.print(',');
                  Serial.print(gy);
Serial.print(',');
        Serial.println(gz);

        delay(sampleDelayMs);
      }
    }
  }
}

// --- Helper functions ---
void openGate() {
   for (int pos = 90; pos <= 180; pos += 2) {
    gateServo.write(pos);
    delay(15);
  }
  Serial.println("Servo opened (90 → 180)");
}

void closeGate() {
   for (int pos = 180; pos >= 90; pos -= 2) {
    gateServo.write(pos);
    delay(15);
  }
    Serial.println("Servo  closed (180 → 90)");
}

void wrongCardFlash() {
  // blink red LED a few times
```

```
  digitalWrite(redLedPin, HIGH);
  digitalWrite(greenLedPin, LOW);

  delay(100);
}

void loop() {
  if (Serial.available()) {
    char cmd = Serial.read();

    if (cmd == '1') {
      // Access granted
         digitalWrite(greenLedPin,
HIGH);
           digitalWrite(redLedPin,
LOW);
      openGate();
    }
    else if (cmd == '0') {
      // System disarmed
         digitalWrite(greenLedPin,
LOW);
           digitalWrite(redLedPin,
HIGH);
      closeGate();
    }
    else if (cmd == 'X') {
      // Wrong card feedback
      wrongCardFlash();
    }
    else if (cmd == 'A') {
          unsigned long start =
millis();
        while (millis() - start <
streamDurationMs) {
          int16_t ax, ay, az, gx,
gy, gz;
          mpu.getMotion6(&ax, &ay,
&az, &gx, &gy, &gz);

                Serial.print(ax);
Serial.print(',');
```

```
  for (int i = 0; i < 3; i++) {
    digitalWrite(redLedPin, HIGH);
    delay(150);
    digitalWrite(redLedPin, LOW);
    delay(150);
  }
    digitalWrite(redLedPin,  HIGH);
// return to idle red ON
   Serial.println("Access denied -
red LED flashed");
}
```

4.2.3. Python Script for Servo and Visualization at PyCharm

A Python script was developed in PyCharm to receive serial data from the Arduino and provide visual or textual feedback. The script displayed RFID scan results and motion verification status in real time. Whenever access was granted, the Python terminal output confirmed successful authentication. When access was denied, it printed an error message, indicating either an invalid card or incorrect gesture.

```
import serial
import time
import math

# single serial port to Arduino (which also provides MPU6050
data)
arduino = serial.Serial("COM7", 9600, timeout=1)
time.sleep(2)

AUTHORIZED_ID = "0013082958"
armed = False
led_on = False

print("System Ready ✅")

def detect_circle():
    """
     Ask Arduino to stream a short burst of MPU data, collect
points,
    and decide whether a circular motion happened.
    Returns True if gesture detected (lenient), False otherwise.
    """
     print("🔄 Requesting samples from Arduino... Move in a circle
now!")

    # ask Arduino to stream (~2 seconds)
    arduino.reset_input_buffer()  # clear old data
    arduino.write(b'A')

    points = []
    start = time.time()
    timeout = 3.0  # safety timeout

    # read until timeout (Arduino will stop streaming after ~2s)
    while time.time() - start < timeout:
        raw = arduino.readline().decode(errors='ignore').strip()
        if not raw:
            continue
        # expect: ax,ay,az,gx,gy,gz
        parts = raw.split(',')
        if len(parts) >= 3:
            try:
                ax = int(parts[0])
                ay = int(parts[1])
                az = int(parts[2])
```

```
                points.append((ax, ay, az))
            except ValueError:
                # skip malformed lines
                continue

    print(f"Collected {len(points)} samples.")

    if len(points) < 8:
                print("⚠️ Not enough motion data — try moving more/stronger.")
        return False

    # Build angles from (ax, ay) and unwrap to follow rotation
       angles = [math.degrees(math.atan2(p[1], p[0])) for p in points]

    # Unwrap angles to avoid -180/180 discontinuity
    unwrapped = [angles[0]]
    for a in angles[1:]:
        prev = unwrapped[-1]
        diff = a - prev
        if diff > 180:
            diff -= 360
        elif diff < -180:
            diff += 360
        unwrapped.append(prev + diff)

      # Compute net rotation and also range (helps with noisy motion)
    net_rotation = unwrapped[-1] - unwrapped[0]
    rotation_range = max(unwrapped) - min(unwrapped)

           print(f"Net rotation: {net_rotation:.1f}°, Range: {rotation_range:.1f}°")

     # LENIENT detection logic (either a decent net rotation OR wide range)
      # Lower thresholds to make detection easier for imperfect circles
    if abs(net_rotation) > 100 or rotation_range > 140:
        print("✅ Circle gesture detected!")
        return True

    # extra check: count direction-consistent large angle deltas
          deltas = [unwrapped[i+1]-unwrapped[i] for i in range(len(unwrapped)-1)]
     large_moves = sum(1 for d in deltas if abs(d) > 10)  # count meaningful moves
    if large_moves >= max(6, len(points)//4):
           print("✅ Motion had multiple consistent movements — accepting as circle.")
        return True

       print("❌ Motion insufficient — try a clearer circular motion.")
    return False



while True:
```

```
    try:
        card = input("Tap card: ").strip()
    except KeyboardInterrupt:
        print("\nExiting.")
        break

    if card == AUTHORIZED_ID:
        armed = not armed

        if armed:
            print("✅ Card Accepted — now do circular motion!")

            if detect_circle():
                print("✅ Gesture OK → LED ON")
                arduino.write(b'1')
                led_on = True
            else:
                print("❌ Wrong gesture — try again")
                armed = False

        else:
            print("🔒 System Disarmed → LED OFF")
            arduino.write(b'0')
            led_on = False

    else:
        print("❌ WRONG CARD")

        arduino.write(b'X')
```

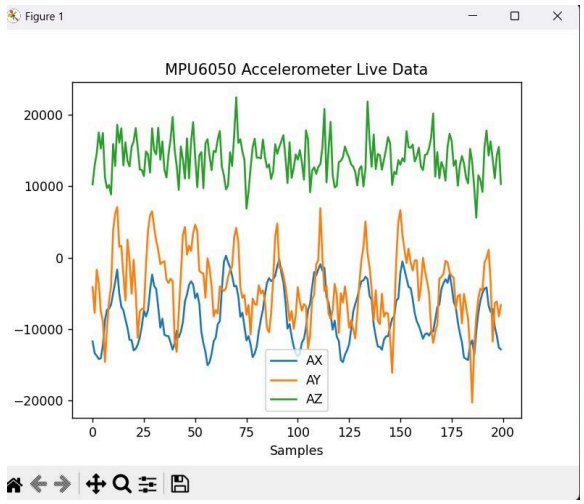
5. Data Collection

5.1 Task 1 – Motion Detection Using MPU6050

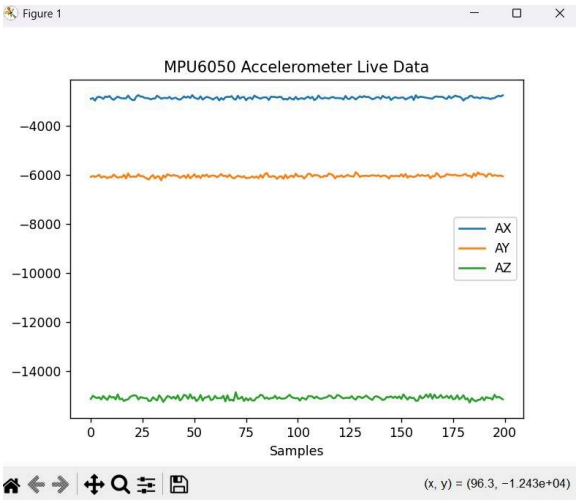
The MPU6050 sensor data (X, Y and Z axis) was collected while performing a circular hand motion. The Arduino sent this data to the Python program, which plotted the movement in real-time using Matplotlib.

Test	Motion Type	Observed Pattern on Graph
1	Circular	Sinusoid pattern
2	No Motion	Line pattern only
3	Random Motion	Scattered pattern

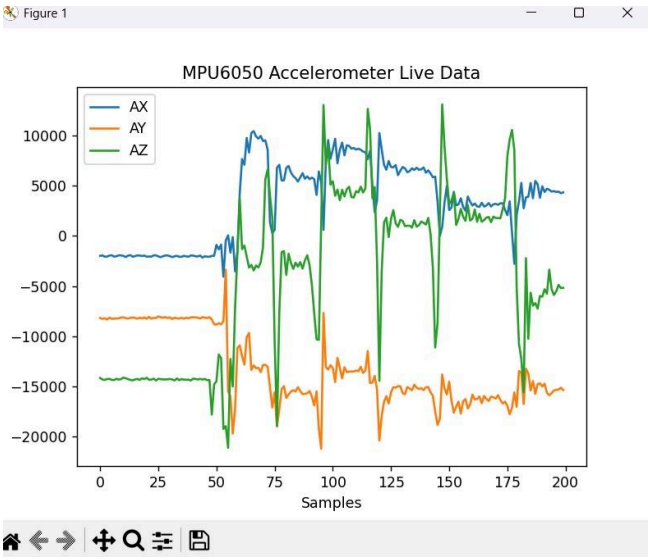
Table 5.1.1 Shows the Data Collection for Task 1



Test 1: Circular



Test 2: No motion



Test 3: Random Motion

Table 5.1.2 Shows the Matplotlib from PyCharm

5.2 Task 2

In this task, the RFID tag was scanned first, followed by the hand motion using the MPU6050. If both the RFID card and motion were correct, the system granted access (servo rotated and green LED ON). If wrong, the red LED turned ON and blinked 3 times.

Test	RFID UID	Motion Type	Access Result	LED Status	Servo Action
1	0013082958 (Authorized)	Circular	Access Granted	Green ON	Rotated 90°
2	0000000000 (Unauthorized)	Random or Circular	Access Denied	Red Blinks 3×	No Movement

Table 5.2.1 Shows the Data Collection for Task 2

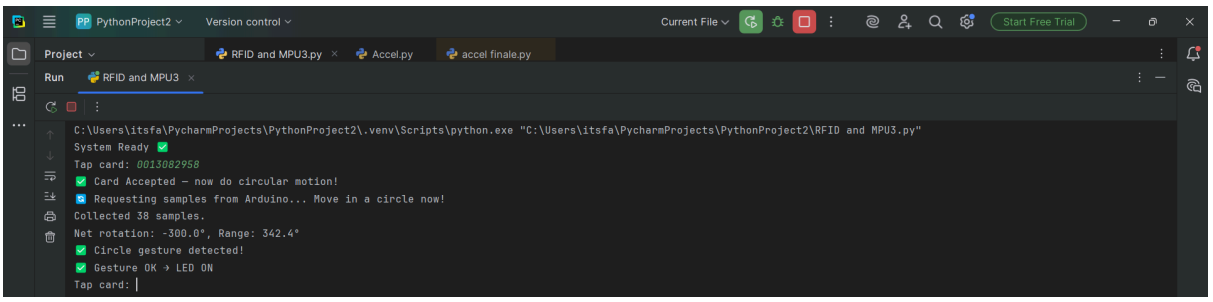


Figure 5.2.1 Shows the Output Visualization from PyCharm for Test 1

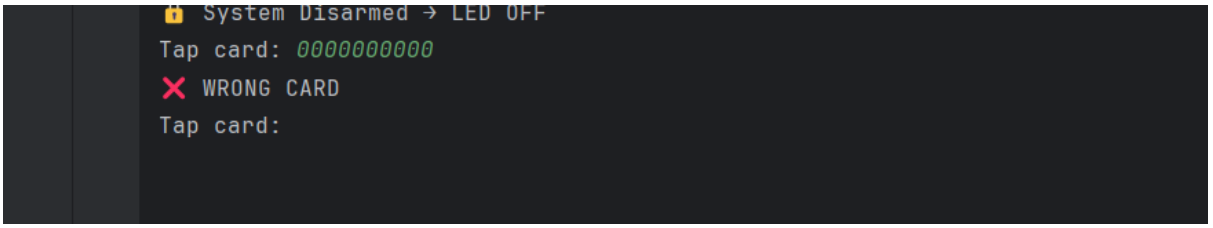


Figure 5.2.2 Shows the Output Visualization from PyCharm for Test 2

6. Data Analysis

6.1 Task 1

The accelerometer data showed how the hand moved over time.

- When a circular motion was performed, the plotted data formed is a distinct fluctuation which visualises the value change over time on the Matplotlib graph.
- The Python code detected the repetitive change in X and Y directions and displayed the message “Circular motion detected.”
- Straight or random movements did not match the circular pattern, so the program correctly showed “No circular motion.”

This shows that the MPU6050 sensor and Python processing worked together to recognize specific motion gestures accurately.

6.2 Task 2

Two types of input were tested:

1. **RFID verification** – checked if the scanned UID matched the list of authorized cards.
2. **Motion verification** – used the same circular motion detection from Task 1.

The system performed serial communication between **Python and Arduino**:

- Python read the UID from Arduino.
- If the UID was authorized, Python asked the user to perform a motion.
- If motion was valid, it sent command **‘A’** to Arduino → servo turned 90°, green LED ON.
- If motion or UID failed, it sent **‘D’** → servo stayed locked, red LED blinked three times.

From the test data, all valid cards combined with the correct circular motion gave an “Access Granted” output, proving that the integrated system worked as intended.

7. Results

7.1 Task 1

During the experiment, real-time accelerometer data along the X and Y axes were successfully captured from the MPU6050 sensor and transmitted to the computer via a serial connection. The data were continuously plotted using Matplotlib to provide a dynamic visualization of the sensor's motion, allowing clear observation of changes in acceleration over time. Through the implemented Python algorithm, the system was also able to analyze the accelerometer readings and effectively detect circular motion gestures, demonstrating its capability to recognize specific movement patterns in real time.

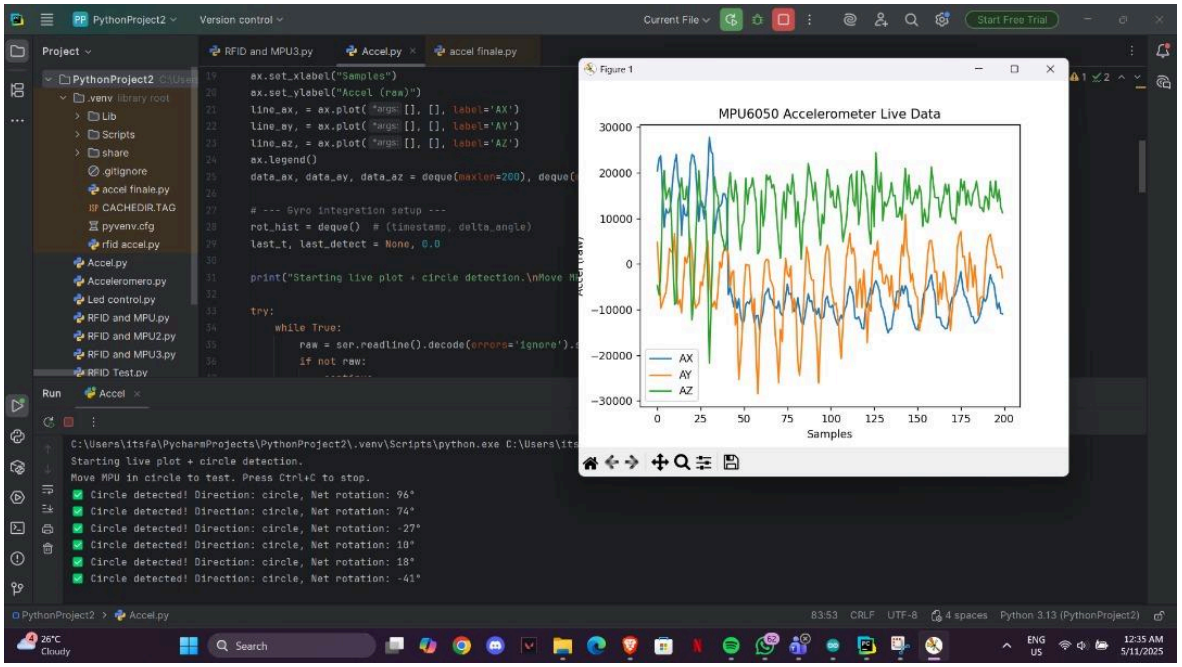


Figure 7.1.1 Shows the full output of the PyCharm for Task 1

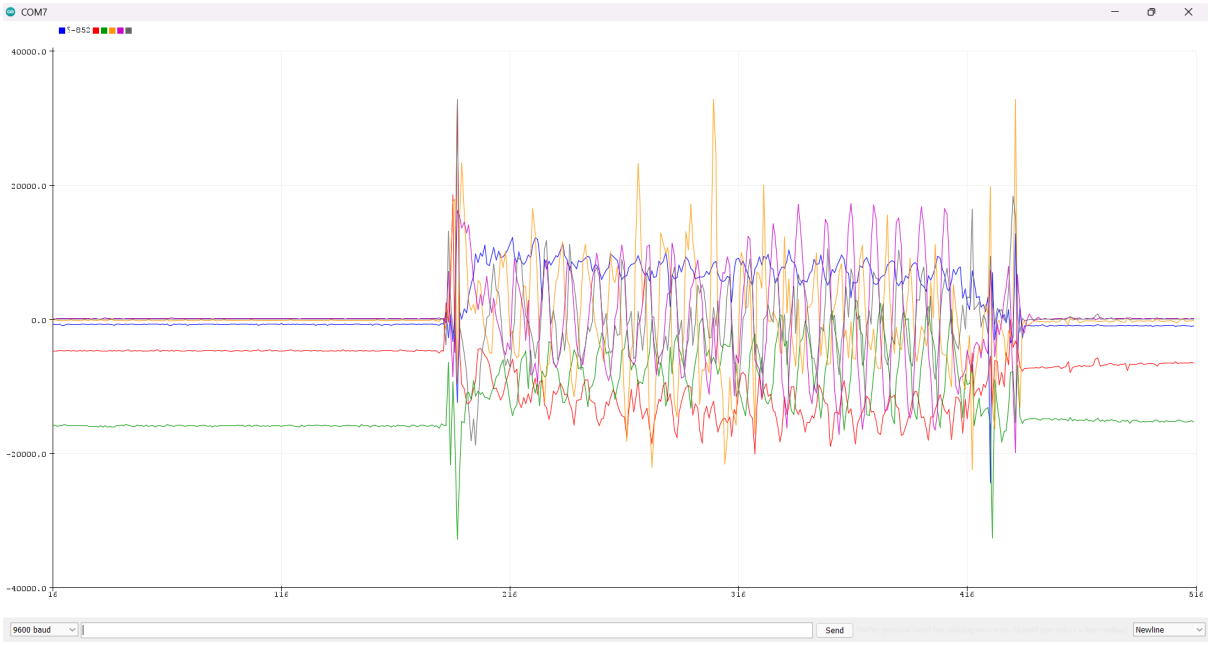


Figure 7.1.2 Shows the Serial Plotter from Arduino IDE

7.2 Task 2

In this experiment, The system worked as intended, combining RFID scanning, motion sensing, and servo control into one smooth process. When an authorized card was tapped, the program asked the user to perform a circular motion. The MPU6050 sensor captured the movement in real time, and once a circular gesture was detected, the green LED turned on and the servo rotated from 90° to 180°, simulating a gate opening. Tapping the same card again returned the servo to its starting position and switched the light back to red, showing the system was locked. If a wrong card was scanned, the red LED flashed to indicate access was denied. Overall, the setup responded reliably and showed how RFID authentication and motion detection can work together for interactive control.

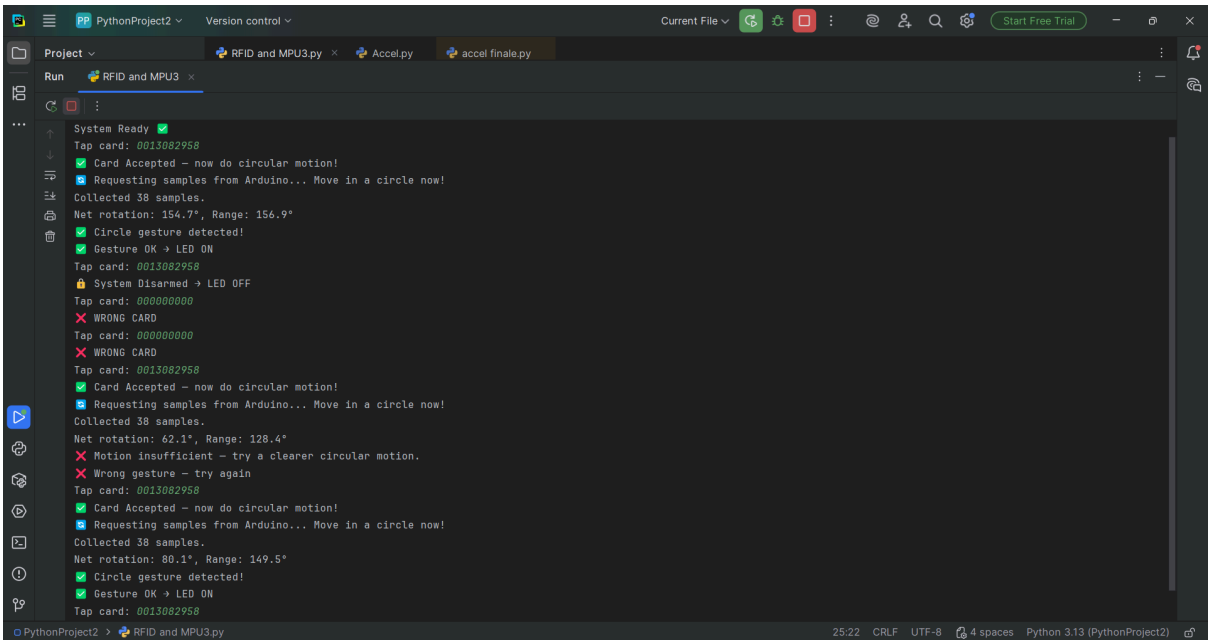


Figure 7.2.1 Shows the Output Visualization from PyCharm

8. Discussion

This project successfully achieved both Task 1 and Task 2 objectives by combining motion tracking with an RFID-based access control system. For Task 1, the MPU6050 sensor was used to capture real-time hand movement data, which was then sent to Python through serial communication. The data was visualized dynamically using Matplotlib, allowing the motion path to be displayed as it happened. This made it possible to observe how the hand moved in space and confirm that the sensor accurately followed changes in direction and acceleration. The system also analyzed these readings to recognize circular hand gestures, demonstrating how motion data can be used for intuitive gesture detection.

In Task 2, this motion recognition feature was integrated into a security setup using an RFID scanner. When a tag was tapped, the system checked whether the card's UID matched the authorized ID. If it did, the user was prompted to perform the circular motion gesture, which served as an additional verification step. Successful completion of the motion triggered the Arduino to turn on the green LED and rotate a servo motor from 90° to 180°, representing an unlocked or open gate. Tapping the same card again returned the servo to its starting position and reactivated the red LED, indicating the system was locked. If an unauthorized card was scanned, the red LED flashed to show access denial.

Overall, the system worked reliably, showing smooth coordination between the RFID module, MPU6050 sensor, and Arduino components. It effectively demonstrated how motion data could enhance access control systems by adding a gesture-based verification layer. This experiment proved that real-time motion tracking and hardware control can be integrated into a simple yet functional automated gate system.

9. Conclusion

The experiment successfully achieved its objectives by integrating real-time motion tracking with RFID-based access control using Arduino and Python. In Task 1, the MPU6050 accelerometer and gyroscope module effectively captured and transmitted hand motion data in real time. The data were dynamically plotted using Matplotlib in Python, allowing for direct visualization of the sensor's movement path. The program's ability to detect circular gestures confirmed the successful implementation of motion recognition and demonstrated the practical use of real-time accelerometer data for gesture-based control.

In Task 2, the system was expanded into a comprehensive access control mechanism that validated user identity through both RFID authentication and motion verification. When an authorized RFID tag was scanned, the system prompted the user to perform a circular hand motion. The captured motion data were analyzed in Python, and when the correct gesture was detected, the system triggered the servo motor to rotate from 90° to 180°, simulating an unlocking action, while the green LED illuminated to indicate authorized access. Scanning the card again caused the servo to return to its initial position and the red LED to turn on, indicating a locked state. For unauthorized RFID tags, the red LED flashed as a visual warning. The consistent and accurate response of the system verified reliable communication between hardware and software components. Overall, the experiment demonstrated an effective integration of sensor technology, data processing, and actuator control to create a functional, intelligent access control system.

10. Recommendations

1. Enhance Motion Detection Accuracy and Stability

To further improve system precision, the motion detection algorithm could incorporate advanced filtering methods such as Kalman or complementary filters. These filters would minimize sensor noise and drift in MPU6050 readings, ensuring smoother data output and more accurate gesture recognition. Additionally, applying data smoothing techniques or adaptive thresholding could help the system distinguish between intentional gestures and small unintended movements, leading to more reliable detection performance.

2. Implement Multi-User RFID Authorization and Data Logging

Expanding the system to support multiple RFID tags would make it more practical for real-world applications, such as office access control, student attendance tracking, or shared facility management. This can be achieved by maintaining a database or file containing a list of authorized UIDs that can be easily updated. Furthermore, integrating a data logging feature would allow the system to record each access attempt — including RFID ID, timestamp, and gesture verification result — enabling administrators to track usage history and enhance system security.

3. Integrate User Feedback and Automation Enhancements

To improve user interaction and usability, additional output devices such as a buzzer or an LCD/OLED display could be incorporated. The buzzer could provide audible feedback for successful or failed attempts, while a display could show status messages like “Access Granted,” “Access Denied,” or “Perform Motion Gesture.” These additions would create a more intuitive interface and make the system easier to use. In addition, integrating wireless communication modules such as Bluetooth or Wi-Fi could enable remote monitoring and control, allowing administrators to manage or update the system without a direct USB connection.

10. References

Last Minute Engineers. (n.d.). *What is RFID? How it works? Interface RC522 RFID module with Arduino.* Retrieved from :https://lastminuteengineers.com/how-rfid-works-rc522-arduino-tutorial/

Random Nerd Tutorials. (n.d.). *Security access using MFRC522 RFID reader with Arduino.* Retrieved from :https://randomnerdtutorials.com/security-access-using-mfrc522-rfid-reader-with-arduino/

Random Nerd Tutorials. (n.d.). *Arduino time attendance system with RFID.* Retrieved from :https://randomnerdtutorials.com/arduino-time-attendance-system-with-rfid/

Instructables. (n.d.). *Arduino + MFRC522 RFID reader.* Retrieved from https://www.instructables.com/Arduino-MFRC522-RFID-READER/

Appendix A - Circuit Diagram

Task 1

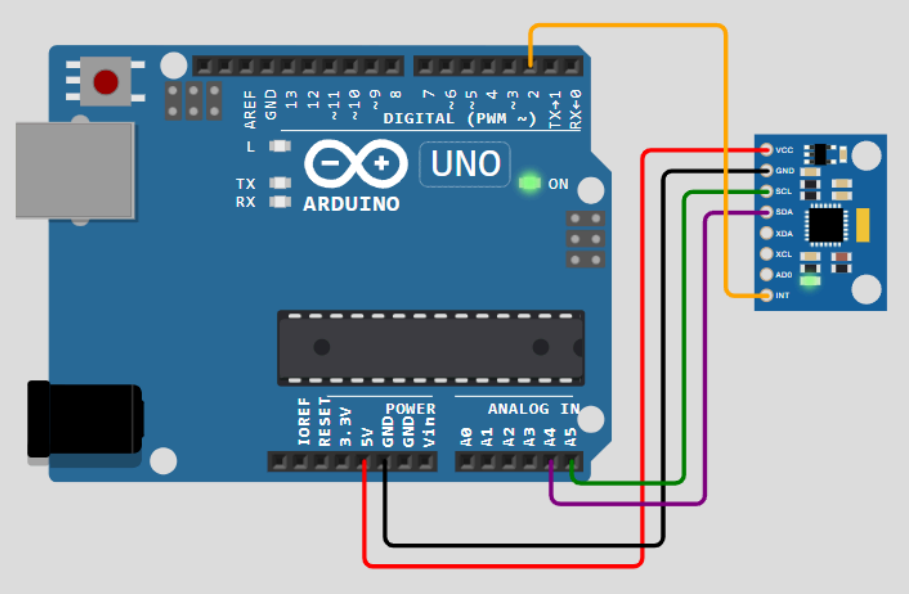


Figure A1 Shows the connection of Task 1 circuit configuration by using Wokwi

Task 2

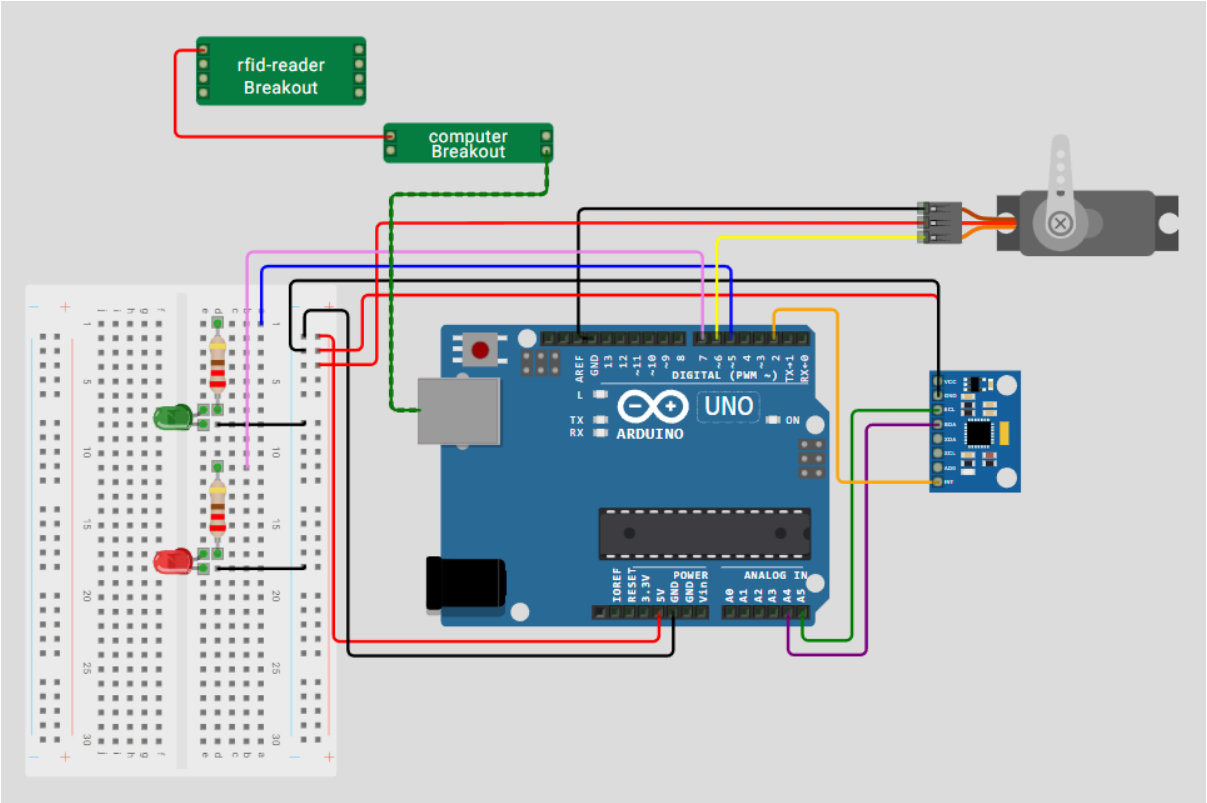


Figure A3 Shows the connection of Task 2 circuit configuration by using Wokwi

Appendix B - Coding Snippets

Task 1

B1 Arduino IDE Coding

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  mpu.initialize();

  if (!mpu.testConnection()) {
    Serial.println("MPU6050 connection failed!");
    while (1);
  }
}

void loop() {
  int16_t ax, ay, az, gx, gy, gz;
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

  Serial.print(ax); Serial.print(",");
  Serial.print(ay); Serial.print(",");
  Serial.print(az); Serial.print(",");
  Serial.print(gx); Serial.print(",");
  Serial.print(gy); Serial.print(",");
  Serial.println(gz);

  delay(8); // ~125Hz -> faster updates, lower delay
}
```

B2 PyCharm Coding

```
import serial, time, math
from collections import deque
import matplotlib.pyplot as plt

# --- Serial connection ---
ser = serial.Serial('COM7', 9600, timeout=1)  # adjust port if needed

# --- Parameters ---
GYRO_SCALE = 131.0  # LSB per °/s for ±250 dps
DEADZONE = 5.0  # ignore small gyro noise
CIRCLE_DEG = 340  # total rotation threshold for detection
WINDOW = 2.0  # seconds of recent data kept
COOLDOWN = 1.5  # seconds between detections

# --- Live plot setup (optional) ---
plt.ion()
fig, ax = plt.subplots()
ax.set_title("MPU6050 Accelerometer Live Data")
ax.set_xlabel("Samples")
ax.set_ylabel("Accel (raw)")
line_ax, = ax.plot([], [], label='AX')
line_ay, = ax.plot([], [], label='AY')
line_az, = ax.plot([], [], label='AZ')
```

```
            # --- Plot accel live ---

data_ax.append(ax_raw)

data_ay.append(ay_raw)

data_az.append(az_raw)

line_ax.set_xdata(range(len(data_ax)))

line_ax.set_ydata(list(data_ax))

line_ay.set_xdata(range(len(data_ay)))

line_ay.set_ydata(list(data_ay))

line_az.set_xdata(range(len(data_az)))

line_az.set_ydata(list(data_az))
            ax.relim()
            ax.autoscale_view()
            fig.canvas.draw()

fig.canvas.flush_events()

                    # --- Gyro integration ---
            now = time.time()
            if last_t is None:
                last_t = now
                continue
            dt = now - last_t
            last_t = now
```

```
        ax.legend()
        data_ax, data_ay, data_az = deque(maxlen=200), deque(maxlen=200), deque(maxlen=200)

        # --- Gyro integration setup ---
        rot_hist = deque()  # (timestamp, delta_angle)
        last_t, last_detect = None, 0.0

        print("Starting live plot + circle detection.\nMove MPU in circle to test. Press Ctrl+C to stop.")

        try:
            while True:
                raw = ser.readline().decode(errors='ignore').strip()
                if not raw:
                    continue

                vals = raw.split(',')
                if len(vals) != 6:
                    continue

                try:
                    ax_raw, ay_raw, az_raw, gx_raw, gy_raw, gz_raw = map(int, vals)
                except ValueError:
                    continue

                gz_dps = gz_raw / GYRO_SCALE
                if abs(gz_dps) < DEADZONE:
                    gz_dps = 0
                dtheta = gz_dps * dt  # incremental rotation in degrees
                rot_hist.append((now, dtheta))

                # drop old samples
                while rot_hist and now - rot_hist[0][0] > WINDOW:
                    rot_hist.popleft()

                # --- Detection ---
                total = sum(d for _, d in rot_hist)
                abs_total = sum(abs(d) for _, d in rot_hist)

                if (abs(total) > CIRCLE_DEG or abs_total > CIRCLE_DEG) and (now - last_detect > COOLDOWN):
                    # direction = "CW" if total < 0 else "CCW"
                    print(f"✅ Circle detected! Direction: circle, Net rotation: {total:.0f}°")
                    last_detect = now
        except KeyboardInterrupt:
            print("\nStopped.")
        finally:
            ser.close()
```

Task 2

B3 Arduino IDE Coding

```
#include <Wire.h>
#include <MPU6050.h>
#include <Servo.h>

MPU6050 mpu;
Servo gateServo;

const int greenLedPin = 5;     // authorized LED
const int servoPin = 6;        // servo motor pin
const int redLedPin = 7;       // unauthorized LED

// Motion streaming config
const unsigned long streamDurationMs = 2000;  // 2 seconds
const unsigned long sampleDelayMs = 50;       // ~20 Hz

void setup() {
  pinMode(greenLedPin, OUTPUT);
  pinMode(redLedPin, OUTPUT);
  gateServo.attach(servoPin);
  gateServo.write(90); // start at closed

  Serial.begin(9600);
  Wire.begin();
  mpu.initialize();
  if (!mpu.testConnection()) {
          Serial.println("MPU6050 connection failed!");
    while (1);
  }
   // Default state: red LED ON (idle/unauthorized)
                Serial.print(ay);
Serial.print(',');
                Serial.print(az);
Serial.print(',');
                Serial.print(gx);
Serial.print(',');
                Serial.print(gy);
Serial.print(',');
        Serial.println(gz);

        delay(sampleDelayMs);
      }
    }
  }
}

// --- Helper functions ---
void openGate() {
   for (int pos = 90; pos <= 180; pos += 2) {
    gateServo.write(pos);
    delay(15);
  }
  Serial.println("Servo opened (90 → 180)");
}

void closeGate() {
   for (int pos = 180; pos >= 90; pos -= 2) {
    gateServo.write(pos);
    delay(15);
  }
    Serial.println("Servo closed (180 → 90)");
}

void wrongCardFlash() {
```

```
  digitalWrite(redLedPin, HIGH);
  digitalWrite(greenLedPin, LOW);

  delay(100);
}

void loop() {
  if (Serial.available()) {
    char cmd = Serial.read();

    if (cmd == '1') {
      // Access granted
         digitalWrite(greenLedPin,
HIGH);
           digitalWrite(redLedPin,
LOW);
      openGate();
    }
    else if (cmd == '0') {
      // System disarmed
         digitalWrite(greenLedPin,
LOW);
           digitalWrite(redLedPin,
HIGH);
      closeGate();
    }
    else if (cmd == 'X') {
      // Wrong card feedback
      wrongCardFlash();
    }
    else if (cmd == 'R') {
          unsigned long start =
millis();
        while (millis() - start <
streamDurationMs) {
          int16_t ax, ay, az, gx,
gy, gz;
          mpu.getMotion6(&ax, &ay,
&az, &gx, &gy, &gz);
Serial.print(ax);
Serial.print(',');
```

```
  // blink red LED a few times
  for (int i = 0; i < 3; i++) {
    digitalWrite(redLedPin, HIGH);
    delay(150);
    digitalWrite(redLedPin, LOW);
    delay(150);
  }
    digitalWrite(redLedPin,  HIGH);
// return to idle red ON
   Serial.println("Access denied -
red LED flashed");
}
```

B4 PyCharm Coding

```
import serial
import time
import math

# single serial port to Arduino (which also provides MPU6050 data)
arduino = serial.Serial("COM7", 9600, timeout=1)
time.sleep(2)

AUTHORIZED_ID = "0013082958"
armed = False
led_on = False

print("System Ready ✅")

def detect_circle():
    """
      Ask Arduino to stream a short burst of MPU data, collect points,
    and decide whether a circular motion happened.
      Returns True if gesture detected (lenient), False otherwise.
    """
     print("🔄 Requesting samples from Arduino... Move in a circle now!")

    # ask Arduino to stream (~2 seconds)
    arduino.reset_input_buffer()  # clear old data
    arduino.write(b'R')

    points = []
    start = time.time()
    timeout = 3.0  # safety timeout

     # read until timeout (Arduino will stop streaming after ~2s)
    while time.time() - start < timeout:
                                                raw       =
```

```
arduino.readline().decode(errors='ignore').strip()
            if not raw:
                continue
            # expect: ax,ay,az,gx,gy,gz
            parts = raw.split(',')
            if len(parts) >= 3:
                try:
                    ax = int(parts[0])
                    ay = int(parts[1])
                    az = int(parts[2])
                    points.append((ax, ay, az))
                except ValueError:
                    # skip malformed lines
                    continue

        print(f"Collected {len(points)} samples.")

        if len(points) < 8:
                print("⚠️ Not enough motion data — try moving more/stronger.")
            return False

          # Build angles from (ax, ay) and unwrap to follow rotation
         angles = [math.degrees(math.atan2(p[1], p[0])) for p in points]

        # Unwrap angles to avoid -180/180 discontinuity
        unwrapped = [angles[0]]
        for a in angles[1:]:
            prev = unwrapped[-1]
            diff = a - prev
            if diff > 180:
                diff -= 360
            elif diff < -180:
                diff += 360
            unwrapped.append(prev + diff)

        # Compute net rotation and also range (helps with noisy motion)
```

```
        net_rotation = unwrapped[-1] - unwrapped[0]
        rotation_range = max(unwrapped) - min(unwrapped)

            print(f"Net rotation: {net_rotation:.1f}°, Range: {rotation_range:.1f}°")

        # LENIENT detection logic (either a decent net rotation OR wide range)
            # Lower thresholds to make detection easier for imperfect circles
        if abs(net_rotation) > 100 or rotation_range > 140:
            print("✅ Circle gesture detected!")
            return True

         # extra check: count direction-consistent large angle deltas
            deltas = [unwrapped[i+1]-unwrapped[i] for i in range(len(unwrapped)-1)]
         large_moves = sum(1 for d in deltas if abs(d) > 10)  # count meaningful moves
        if large_moves >= max(6, len(points)//4):
            print("✅ Motion had multiple consistent movements — accepting as circle.")
            return True

         print("❌ Motion insufficient — try a clearer circular motion.")
        return False



    while True:
        try:
            card = input("Tap card: ").strip()
        except KeyboardInterrupt:
            print("\nExiting.")
            break

        if card == AUTHORIZED_ID:
            armed = not armed
```

```
            if armed:
                    print("✅ Card Accepted — now do circular
motion!")

                if detect_circle():
                    print("✅ Gesture OK → LED ON")
                    arduino.write(b'1')
                    led_on = True
                else:
                    print("❌ Wrong gesture — try again")
                    armed = False

            else:
                print("🔒 System Disarmed → LED OFF")
                arduino.write(b'0')
                led_on = False

        else:
            print("❌ WRONG CARD")

            arduino.write(b'X')
```

Appendix C - Results

Task 1

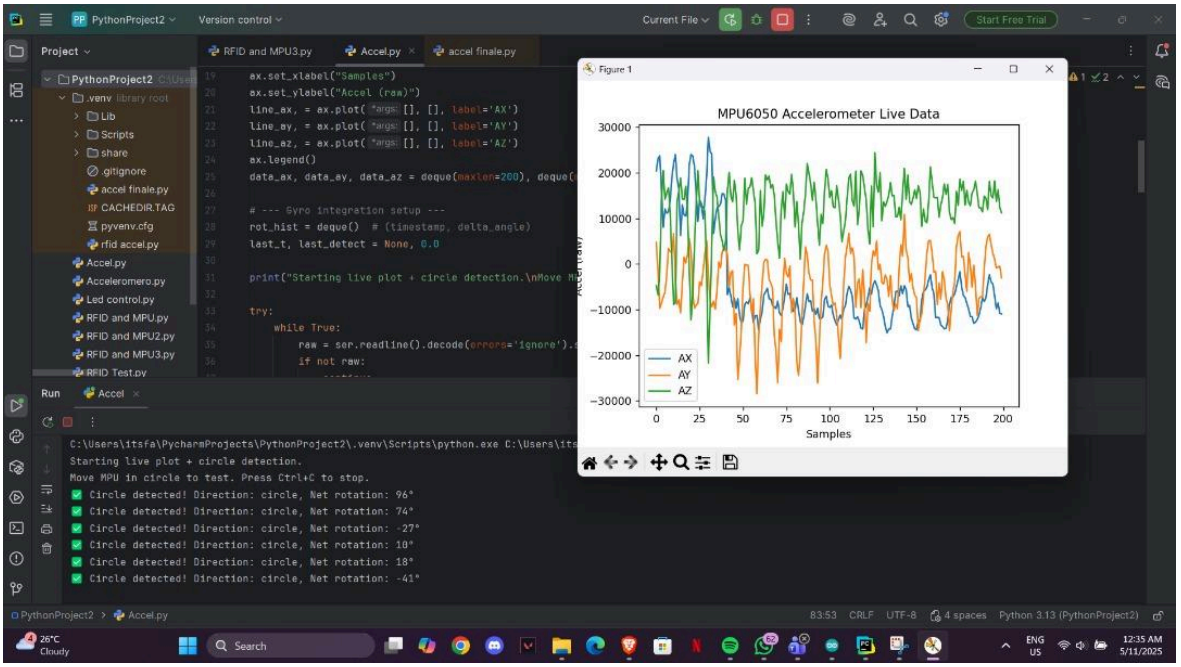


Figure C1 Shows the full output of the PyCharm for Task 1

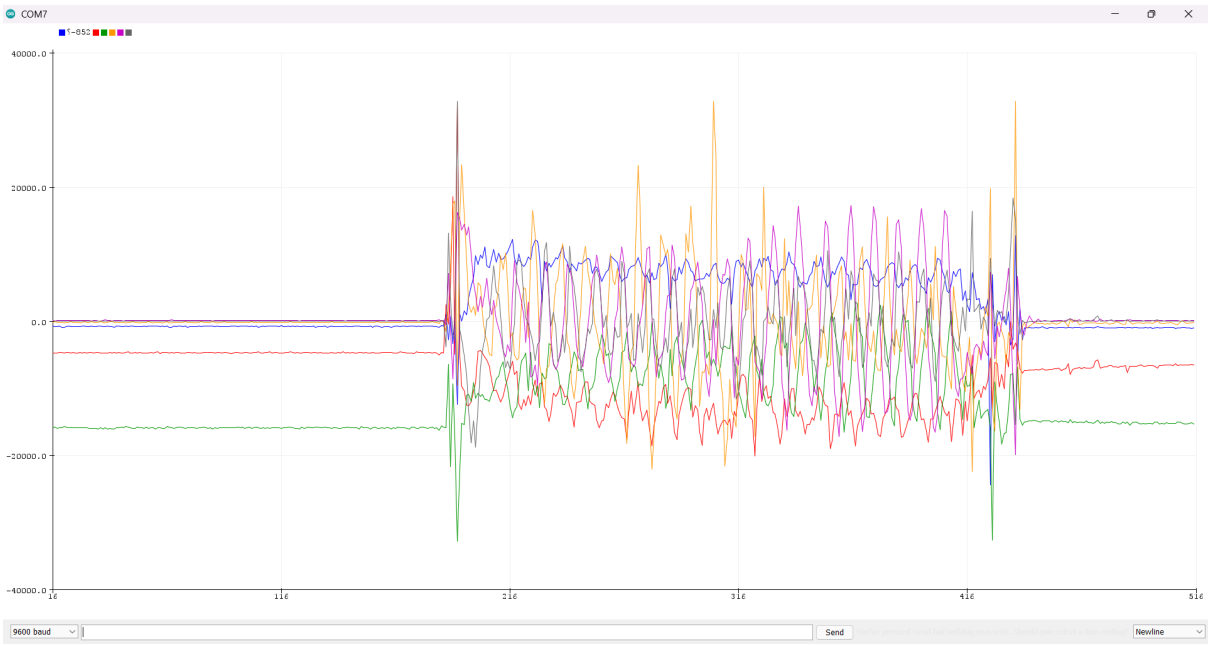


Figure C2 Shows the Serial Plotter from Arduino IDE

Task 2

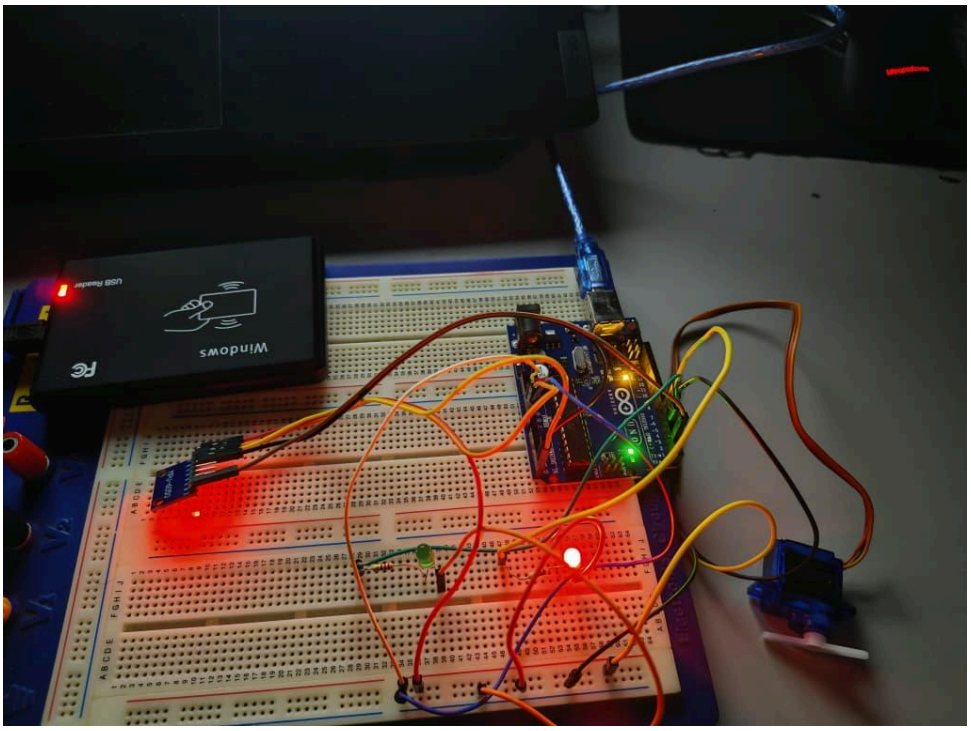


Figure C3 Before tapping the rfid card

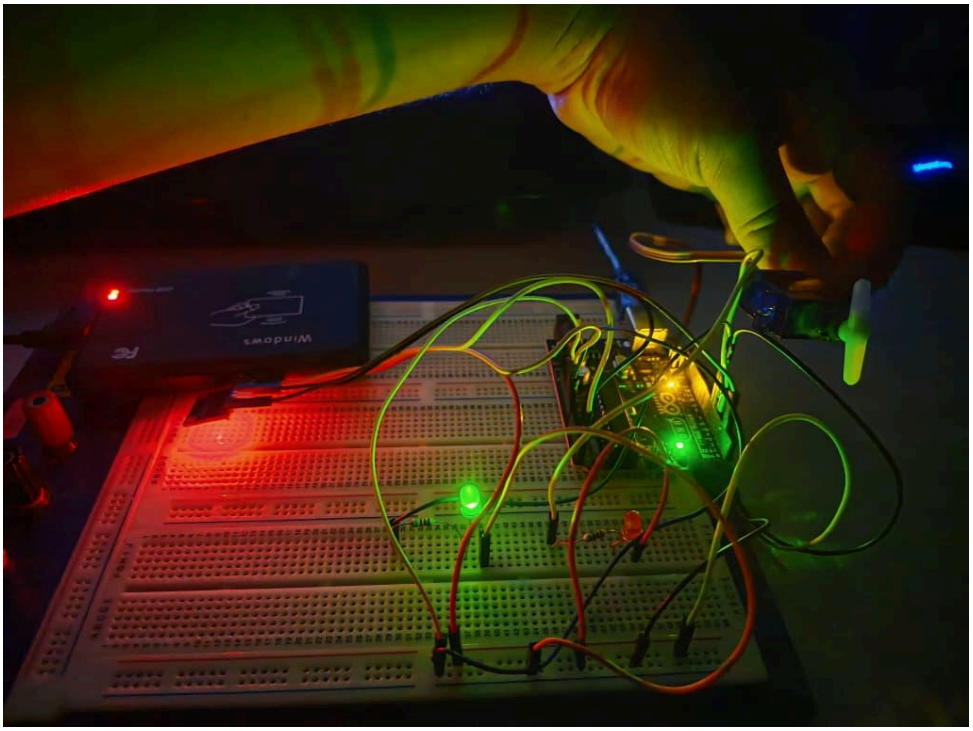


Figure C4 After tapping the rfid card and moved MPU 6050 in circular motion

Acknowledgments

We would like to express our sincere gratitude to our lecturer and lab instructor for their continuous guidance, clear explanations, and assistance with experimental setups throughout this experiment. Their detailed instructions, along with the necessary files provided, helped us understand how to properly set up the Arduino UNO microcontroller, PyCharm, and serial communication, and gave us the confidence to carry out the tasks effectively. We also extend our heartfelt appreciation to the lab assistants, whose patience and support in setting up the equipment, troubleshooting circuits, and clarifying procedural steps were instrumental in the smooth progress of the experiment.

Special thanks are also due to our team members, who contributed significantly to every stage of this project, including circuit assembly, coding, data collection, analysis, and documentation. Their collaboration, dedication, and teamwork ensured that each task was completed accurately and efficiently. We also sincerely appreciate our classmates and lab partners for their cooperation, ideas, and teamwork during the experiment. Together, we troubleshot connection issues, shared components, and tested the code to ensure everything functioned as expected.

Overall, the success of this experiment would not have been possible without the guidance and support of both our instructors and lab partners, and we are truly grateful for their contributions.

Certificate of Originality and Authenticity

This is to certify that we are **responsible** for the work submitted in this report, that **the original work** is our own except as specific in references and acknowledgement, and that the original work contained herein have not been untaken or done by unspecified sources or a person.

We hereby certify that this report has **not been done by only one individual** and **all of us have contributed to the report**. The length of contribution to the reports by each individual is noted within this certificate.

We also hereby certify that we have **read** and **understand** the content of the total report and no further improvement on the reports is needed from any of the individual's contributors to the report.

We, therefore, agreed unanimously that this report shall be submitted for **marking** and this **final printed report** has been **verified by us**.

Signature: 	Read	/
	Understand	/
Name: Muhammad Fahim Bin Ahmad Norisham	Agree	/
Matric Number: 2312369		
Signature: 	Read	/
Name: Muhammad Zamarul Azim Bin Zulkhibri	Understand	/
Matric Number: 2312451	Agree	/
Signature: 	Read	/
Name: Muhammad Hazimuddin Bin Fairuz	Understand	/
Matric Number: 2312479	Agree	/