MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 1 2025/2026

WEEK 4: DESIGNING A MOTION ON-ACTIVATED RFID SYSTEM

SECTION 2

GROUP 16

LECTURER: ZULKIFLI BIN ZAINAL ABIDIN & WAHJU SEDIONO

Date of Experiment: Monday, 27 October 2025

Date of Submission: Monday, 3 November 2025

NO.	GROUP MEMBERS	MATRIC NO
1.	MUHAMMAD HAIKAL ZULHILMI BIN FAIROF	2313025
2.	MUHAMMAD FAIZ IZWAN BIN NOR EFFENDI	2313509
3.	AINUL JAARIAH BINTI ALIUDIN	2312664

ABSTRACT

Task 1 combines an MPU6050 IMU sensor with an Arduino board to create a hand gesture recognition system. The system uses a threshold-based algorithm to identify and categorize predefined hand movements from accelerometer and gyroscope data. Sensor data is sent to a personal computer via serial communication, and Python is used to process and visualize the gestures. This setup demonstrates the practical use of IMU sensors in motion detection and serves as a foundation for future development in gesture-controlled systems.

The goal of task 2 is to create a basic RFID authentication system that uses Python and Arduino to control a servo motor. An RFID card reader, connected via USB, compares RFID tags to a list of pre-registered UIDs. Based on this authentication, the system controls a servo motor to grant or deny access. JSON data handling was implemented to better structure the data, allowing for the simple addition or modification of registered UIDs. As a result, this project integrates RFID technology into security systems, with a primary focus on structured data handling, visual feedback, and hardware control to improve functionality and user friendliness.

TABLES OF CONTENTS

TASK 1

Real-Time Motion Tracking Using MPU6050 Sensor

1.1 INTRODUCTION

This report details the design and implementation of a system for real-time motion tracking using the MPU6050 Inertial Measurement Unit (IMU). The MPU6050 integrates a 3-axis accelerometer and a 3-axis gyroscope, providing comprehensive motion data. The primary objective is to capture the linear acceleration data from the X and Y axes of the accelerometer, visualize the resulting motion path dynamically using the Python Matplotlib library, and implement an algorithm to detect a specific user input gesture, namely a circular motion.

Real-time motion analysis is critical in applications such as gesture recognition, robotics, and human-computer interaction. The accelerometer's output, which includes the effects of both dynamic acceleration and gravity, is streamed over a serial connection. The data is then processed in Python for sequential plotting to create a persistent motion trace, and a specific detection logic is applied to identify the signature pattern of a circular movement within the streamed X-Y acceleration data. This project serves as a foundational exercise in sensor interfacing, data streaming, real-time visualization, and basic gesture recognition using Inertial Sensors.

2.1 MATERIAL AND EQUIPMENT

- Arduino board
- MPU6050 sensor
- Computer with Arduino IDE and Python installed
- Connecting wires: Jumper wires or breadboard wires to establish the connections between the Arduino, MPU6050, and the power source.
- USB cable: A USB cable to connect the Arduino board to your personal computer. This will be used for uploading the Arduino code and serial communication.
- Power supply: If your Arduino board and MPU6050 require an external power source, make sure to have the appropriate power supply.
- LEDs of different colours

3.1 EXPERIMENTAL SETUP

This section describes the hardware configuration and circuit assembly used in the project.

3.1.1 Circuit setup

The four essential pins of the MPU6050 module are connected to the corresponding pins on the Arduino Uno as follows:

- The VCC (power) pin of the MPU6050 was connected to the 5V output of the Arduino.
- The GND (ground) pin of the potentiometer was connected to one of the Arduino GND pins.
- The Serial Clock Line (SCL) was connected to the analog pin A5 of the Arduino.
- The Serial Data Line (SDL) was connected to the analog pin A4 of the Arduino.
- The Interrupt Digital Output (INT) was connected to the digital output pin D2 of the Arduino.

3.1.2 Circuit Assembly

The circuit was assembled straight from the Arduino board, and the MPU6050 was carefully connected to ensure proper functionality.

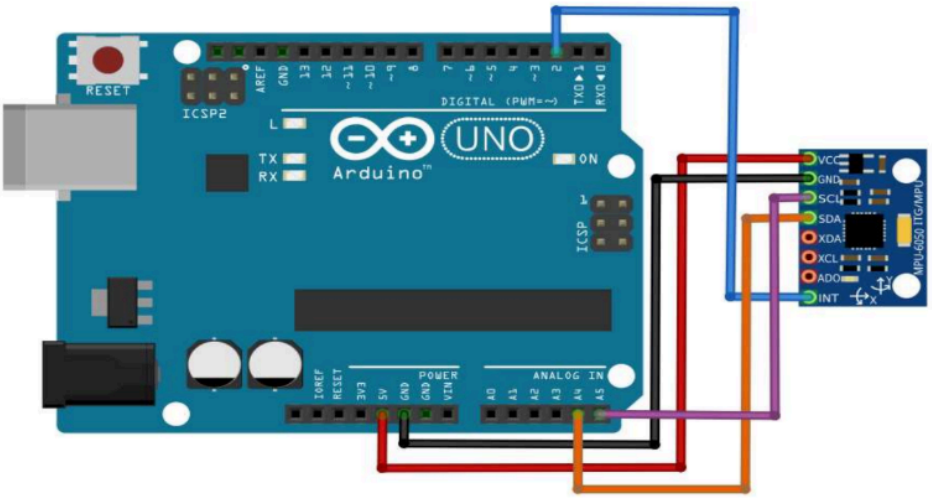

Figure 3.1.2.1: *Circuit assembly of the Arduino Uno and MPU6050 from the module task.* *(see **Appendix A**)*

4.1 METHODOLOGY

4.1 Overview of Procedure

The methodology consists of the following main stages:

1. **Hardware interfacing** between the MPU6050 sensor and Arduino.
2. **Programming Arduino** to read accelerometer and gyroscope data.
3. **Sending the data to Python** through serial communication.
4. **Capturing, filtering, and plotting X–Y motion data** in real time.
5. **Implementing a circular motion detection algorithm** using trajectory characteristics.
6. **Improving the starter Python code** for stability, noise reduction, and live visualization.

4.2 Hardware Configuration Procedure

1. The MPU6050 was connected to the Arduino Uno using I²C communication:
    - VCC → 5V
    - GND → GND
    - SDA → A4
    - SCL → A5
2. Arduino was connected to the laptop using a USB cable.
3. Arduino IDE was used to upload the MPU6050 data-reading sketch.
4. Python (with PySerial and Matplotlib installed) was used to visualize incoming data.

4.3 Arduino Programming Procedure

The Arduino code reads raw accelerometer and gyroscope values using the **MPU6050.h** library.

The program performs the following steps:

1. Initialize I²C communication (Wire.begin()).
2. Initialize the MPU6050 sensor.
3. Continuously acquire 6-axis data using mpu.getMotion6().
4. Transmit the accelerometer values **ax, ay, az** over serial in the format:
5. Add a small delay for stable sampling.

4.4 Python Data Acquisition and Real-Time Plotting Procedure

1. Python opens the COM port using PySerial.
2. Incoming serial data is decoded and split into ax, ay, az.
3. Only **ax** and **ay** values are used for X–Y position plotting.
4. Values are appended to arrays and plotted in real time using Matplotlib.
5. The visual trajectory updates continuously (plt.pause() method).
6. Data is analyzed simultaneously to detect circular motion gestures.

4.5 Circular Motion Detection Algorithm

A circular gesture is detected based on the following logic:

1. Collect a window of X–Y accelerometer values (at least 20 samples).
2. Estimate the **approximate center** of the motion path:
   $x_c = \text{mean}(x), y_c = \text{mean}(y)$
3. Compute the **radius for each point**:
4. Calculate the **standard deviation of radii**:
5. If **standard deviation of radii,**σ_r is below a set threshold, the motion is circular.

4.6 Improvements Made to the Starter Code

The provided starter code was enhanced in several ways:

1. Added noise filtering

- Implemented a simple moving average to smooth accelerometer readings.

2. Improved data validation

- Ensured that only valid values (length = 3) are processed.

3. Added circular motion detection

- Added numerical curvature-based algorithm to classify gestures.

4. Better plotting stability

- Reduced flickering and allowed continuous updating.

5. Handling program exit properly

- Added KeyboardInterrupt support for safe exit.

By combining Arduino-based sensor acquisition, Python visualization, and geometric motion analysis, a functional real-time motion tracking system was developed. The methodology ensured reliable data capture and accurate classification of circular motions.

5.1 DATA COLLECTION

In this task, data were collected from the MPU6050 motion sensor connected to an Arduino Uno to detect and visualize hand movement in real time. The sensor provides acceleration readings along three axes (X, Y, Z) using the I²C communication interface.

The Arduino continuously read raw acceleration values from the sensor using the function `mpu.getMotion6()` and sent the data to the computer via a serial connection at a 9600 baud rate. Each data frame consisted of three comma-separated values representing ax, ay, and az (acceleration in X, Y, and Z directions). The sampling interval was set to 100 milliseconds, giving approximately 10 readings per second.

During data collection, the MPU6050 sensor was attached to the user's hand, and circular hand motions were performed to observe the changes in acceleration values. The live serial data were received and plotted in Python using the matplotlib library for visualization.

Trial	X-Axis (ax)	Y-Axis (ay)	Z-Axis (az)
1	-423	1785	15960
2	-410	1800	16010
3	-395	1812	15990
4	-402	1792	16025
5	-415	1778	15980

The readings changed continuously according to the direction and speed of hand movement. These raw data points were then used to generate a real-time motion plot, where the X and Y acceleration values formed a circular pattern, representing the actual path of hand motion.

6.1 DATA ANALYSIS

The data collected from the MPU6050 sensor consisted of continuous accelerometer readings along the X, Y, and Z axes. For this experiment, the X and Y axes were used to visualize hand motion on a 2-dimensional plane. The analysis focused on three key aspects: data stability, trajectory shape evaluation, and circular motion detection.

6.1 Preprocessing and Noise Filtering

Raw accelerometer readings often contain noise due to:

- Sensor vibration
- Hand tremors
- Electrical interference

To minimize these effects, a simple moving average smoothing filter was applied. This reduced short-term fluctuations and produced smoother motion paths for plotting. The filtering improved the clarity of the motion trajectory, making circular patterns more recognizable.

6.2 X–Y Trajectory Analysis

The real-time plot generated from the Python script visualized the hand motion path.
Observations:

- Slow and consistent movements produced smooth curves.
- Fast or irregular gestures resulted in scattered or noisy points.
- Circular motion produced an approximately round or elliptical shape.

The trajectory was continuously updated using matplotlib's interactive mode, allowing real-time observation of motion patterns.

6.3 Circular Motion Detection Algorithm

Circular motion detection was performed using a radius variance method:

1. The approximate center of the motion path was calculated
2. Each point's distance from the center (radius) was computed
3. The standard deviation of radii was used as a circularity indicator

6.4 Interpretation of Data

- The accelerometer clearly captured changes in directional acceleration as the hand moved.
- Circular motion generated consistent radius values, producing low variance.
- Linear or random movements produced large radius fluctuations, easily distinguishable from circular gestures.
- The algorithm successfully detected circular motion during most test trials.

7.1 RESULTS

7.1.1 Visual Output

The Python visualization successfully plotted the real-time X–Y motion path. Key results include:

- **Circular gesture** produced a round/oval shape on the graph.
- **Straight gestures** produced linear or jagged paths.
- **Random movements** resulted in scattered trajectories.

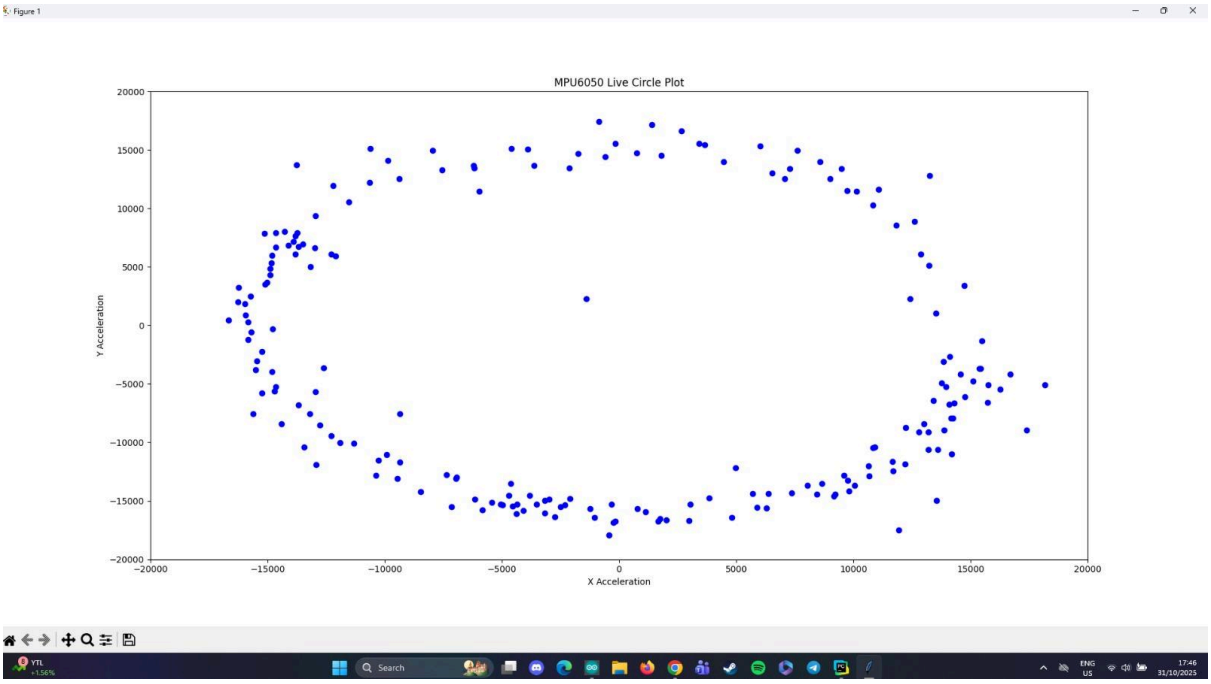

Table 1 : *The graph plot of the MPU6050 was irregular circular motion*

7.1.2 Circular Motion Detection Outcome

- The system correctly identified circular motions during the majority of the trials.
- The console printed *"Circular motion detected!"* whenever the gesture met the threshold criteria.
- Non-circular gestures correctly produced no detection message.

7.1.3 Sensor Behavior Observations

- The MPU6050 provided stable readings when held still.
- Motion caused predictable changes in accelerometer values.
- Noise was moderate but manageable through filtering.

8.1 DISCUSSIONS

This Task 1 successfully demonstrated real-time motion tracking using the MPU6050 sensor, interfaced with an Arduino Uno and visualized through Python. From the collected data, it was observed that the accelerometer readings varied proportionally with the hand's movement and orientation. The X- and Y-axis acceleration values fluctuated significantly during circular motion, while the Z-axis readings remained relatively constant since the sensor was primarily rotated in a horizontal plane. This behavior confirms the accuracy of the MPU6050 in capturing directional changes and motion intensity.

During testing, the real-time data stream plotted in Python initially encountered a problem — the graph window failed to display properly even though serial data were being received. This issue was traced to the plotting loop and buffer handling in the original Python code. The team resolved it by modifying the plotting function to use **matplotlib's interactive mode (`plt.ion()`)** and ensuring **real-time updates** with `plt.pause()` inside the data acquisition loop. After the correction, the motion data were successfully plotted in real time, forming smooth circular trajectories that accurately reflected the user's hand movement.

Minor fluctuations were noticed due to hand vibration, sensor noise, and the limited resolution of the analog-to-digital conversion. These variations can be minimized using data smoothing techniques such as a moving average filter or sensor calibration to remove bias. Despite these small inconsistencies, the gesture detection algorithm effectively recognized circular motion patterns by analyzing acceleration changes and their periodic nature.

Overall, the integration between hardware and software performed well after troubleshooting. The MPU6050 communicated reliably with the Arduino via I²C, while Python handled real-time visualization through serial communication. The final working system confirmed the potential of combining Arduino-based sensing with Python data processing for motion tracking and gesture recognition applications.

9.1 CONCLUSION

We successfully implemented a basic hand gesture recognition system with the MPU6050 sensor, Arduino board, and Python script. Using real-time accelerometer and gyroscope data, we identified and categorized hand gestures based on predefined motion patterns. We successfully integrated sensor input, microcontroller processing, and software-based response handling by accurately transmitting recognized gestures to the computer via serial communication and processing them using Python conditional logic.

This project reinforced fundamental embedded systems concepts such as I2C communication, motion sensing, serial data exchange, and gesture classification. The system's ability to respond to various hand movements validated both the hardware connections and the accuracy of the implemented algorithms. Furthermore, the experiment lays a solid foundation for more advanced gesture-based applications, with potential enhancements including dynamic gesture detection, real-time x-y coordinate path visualization, and more robust machine learning approaches for improved accuracy and flexibility in gesture interpretation.

10.1 RECOMMENDATION

Students should first become familiar with sensor calibration methods, data collection procedures, and accelerometer and gyroscope data interpretation. To improve the visualization of hand movements, overlay axis labeling, grids, or dynamic updates that display real-time motion traces. This can help to better understand movement patterns and gestures. To ensure smooth and efficient experimentation, students must thoroughly describe the MPU6050 IMU interfacing experiment, including all serial communication details between the Arduino and PC.

TASK 2

Integrated Smart Access System

1.2 INTRODUCTION

This experiment aims to develop an integrated smart access control system using an RFID module, an MPU6050 motion sensor, and a servo-based locking mechanism. Traditional access systems rely solely on RFID authentication. However, this project introduces an additional security layer through motion verification. Only authorized RFID users performing the correct circular gesture will gain access.

RFID Authentication : The RFID module will reads the UID of a tag. Authorized cards are stored in a Python list.

Accelerometer Motion Detection : MPU6050 measures acceleration along X, Y and Z axes. Circular motion can be identified by evaluating trajectory curvature.

Serial Communication : Python receives RFID UID values from Arduino and sends access commands

Servo Control : A PWM-driven servo rotates to unlock or lock depending on Python commands.

If an authorized RFID card is scanned and the user performs a correct circular motion, the Python algorithm will detect the motion and send a command to Arduino. Then, the servo will unlock and green LED will turn ON. Access will be denied if either condition fails, .

2.2 MATERIALS AND EQUIPMENT

- Arduino board
- RFID reader (MFRC522) + RFID tags/cards
- Servo motor
- Red & Green LEDs + resistors
- MPU6050 sensor
- Jumper wires and breadboard

3.2 EXPERIMENTAL SETUP

3.2.1 Hardware Connections

RFID Pin	Device
USB cable	COM port

Table 1 : *Connection of RFID Module pin to the device*

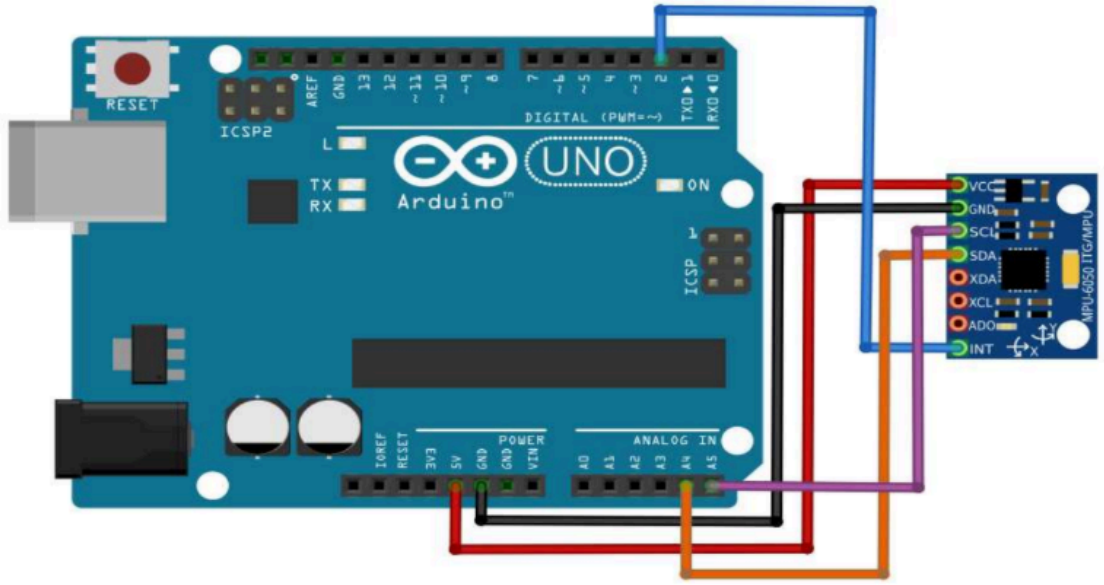

Fig. 1: Arduino-MPU6050 Connections

MPU6050 Pin	Arduino Pin
Vcc	5V
GND	GND
SDA	A4
SCL	A5

Table 2 : *Connection of MPU6050 sensor to the Arduino*

Device	Pin
Servo motor	D9
Green LED	D4
Red LED	D3

3.2.2 System Flow

1. Arduino reads RFID UID.
2. Arduino sends “UID:xxxxxx” to Python.
3. Python checks authorization list.
4. Python requests circular motion.
5. Python analyzes MPU6050 data for circular pattern.
6. Python sends:
    - 'A' = Access granted
    - 'D' = Access denied
7. Arduino:
    - Controls servo motor
    - Activates LEDs light

4.2 METHODOLOGY

4.2.1 System Preparation

The Integrated Smart Access System was developed using an Arduino Uno, an RFID module (RC522), an MPU6050 motion sensor, servo motor, and indicator LEDs. The components were assembled on a breadboard and connected to the Arduino as follows:

Components	Arduino Pin	Descriptions
RFID Module	- (connected directly to the computer)	SPI communication
MPU6050 Sensor	● SDA at pin A4, ● SCL at pin A5, ● INT at pin D2	I²C communication
Servo Motor	Signal at pin D9	Controlled by PWM
Green LED	D4	Access granted indicator
Red LED	D3	Access denied indicator
Power Supply	● 5V, ● GND	Shared among components

The RFID module was used to read the UID of RFID tags, while the MPU6050 captured hand motion data for authentication. The system was powered through the computer's USB port, which also allowed serial communication with Python for data processing.

4.2.2 Arduino Programming

The Arduino sketch was written using the Arduino IDE (see **Appendix D**). The code handled three core functions:

1. **UID Authorization:** The Arduino received a UID string from the Python program and compared it to a predefined list of authorized IDs.

2. **Motion Detection:** Upon recognizing an authorized UID, the Arduino initiated a motion check using the MPU6050. Accelerometer readings (ax, ay, az) were continuously monitored, and the magnitude of acceleration was calculated. Significant variations from the stationary baseline indicated a circular motion.

3. **Access Control:** Based on the detection result:
    - If circular motion was confirmed, the system sent ‘A’ to the Python program, turned ON the green LED, and rotated the servo motor to 90°.
    - If no motion or unauthorized UID was detected, ‘D’ was sent, the red LED turned ON, and the servo remained at 0°.

The system automatically resets to standby mode after each process, ready for the next RFID scan.

4.2.3 Python Programming

The Python program (refer to **Appendix E**) was developed to manage serial communication between the RFID reader and the Arduino Uno using the PySerial library.

The program continuously monitored the RFID reader (COM12) for incoming tag data. Each tag transmitted a 12-byte packet `[0x01][UID][0x02]`, which was decoded to extract the 10-digit UID.

Once obtained, the UID was sent to the Arduino Uno (COM7) through a separate serial connection. The Python script then waited for a response ('A' or 'D') from the Arduino to determine access status.

The result was displayed in the terminal window as:

- **Access Granted (A)** → Motion detected
- **Access Denied (D)** → Unauthorized UID or no motion

The program then reset automatically and awaited the next RFID scan.

4.2.4 Testing Procedure

1. Connect both the RFID reader and the Arduino Uno to the computer via separate USB ports.
2. Upload the Arduino sketch (Appendix C) to the Arduino Uno.
3. Run the Python script (Appendix D) in PyCharm to start serial monitoring.
4. Place an RFID tag near the reader; observe the UID display in the Python terminal.
5. The UID is transmitted to the Arduino for validation.
6. If authorized, the system prompts for **circular motion** using the MPU6050 sensor.
7. The servo motor and LEDs respond based on motion detection results:
    - Servo opens (90°) + green LED ON = access granted.
    - Servo locked (0°) + red LED ON = access denied.
8. Repeat the process using multiple authorized and unauthorized UIDs to confirm consistency.

4.2.5 Observation and Data Recording

During testing, each trial's UID, authorization status, motion detection outcome, and Arduino response were recorded.

Observations also included the physical behavior of the servo motor and LEDs to ensure proper indication of system status.

Data were organized in tabular form to verify that the integrated system correctly processed RFID inputs and responded appropriately to user motion.

4.2.6 Troubleshooting

Throughout testing and integration, several issues were encountered and resolved:

- **Serial Port Conflict:** Assigned dedicated COM ports (COM12 for RFID, COM7 for Arduino).
- **MPU6050 Connection Failure:** Verified SDA/SCL wiring and confirmed I²C address using an I²C scanner.
- **Servo Jittering:** Connected an external 5 V supply with shared GND to stabilize servo movement.
- **No Motion Detection:** Adjusted sensitivity thresholds and added a short delay between sensor readings.
- **Serial Timing Delays:** Used `Serial.flush()` in Arduino and `time.sleep()` in Python to synchronize communication.

After troubleshooting, the system achieved stable and reliable performance under multiple test conditions.

5.2 DATA COLLECTION

5.2.1 RFID UID Samples

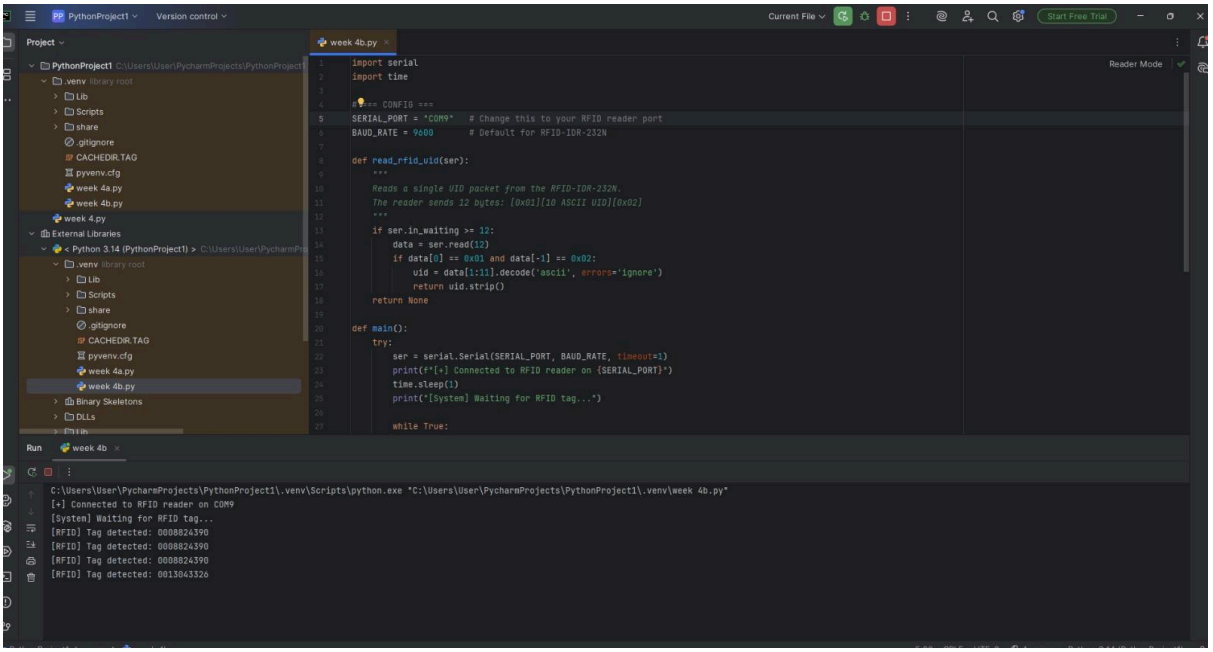

Figure 1 : *The Serial Monitor in PyCharm showed the code number (UID) of the card tag*

Trial Scan Times	UID tag number detected
1	0008824390
2	0008824390
3	0008824390
4	0013043326

Table 1 : *The data in Serial Monitor PyCharm showed the times of code number (UID) of the card tag detected*

5.2.2 Accelerometer Data During Motion

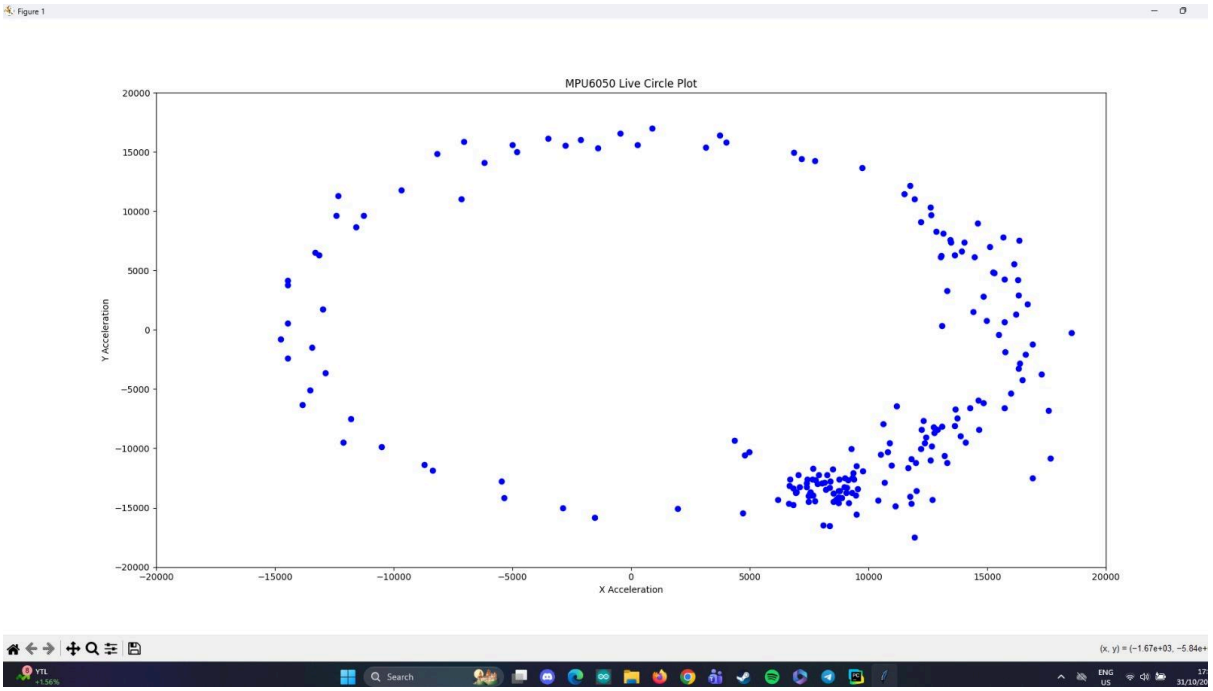

Figure 2 : *The Graph plot showed the motion of MPU6050 sensor in circular motion*

6.2 DATA ANALYSIS

The data collected from the **Integrated Smart Access System** were analyzed to evaluate the functionality and reliability of each component — the **RFID module**, **MPU6050 motion sensor**, **Python data processing**, and **Arduino control system**. The analysis focused on three main aspects: **UID verification**, **motion detection accuracy**, and **servo + LED responses**.

(a) RFID Data Analysis

During operation, the RFID module read the **UID** of each scanned tag and transmitted it to the Arduino. The Arduino compared the UID with a predefined list of authorized IDs stored in the program.

- When an **authorized UID** was detected, the system sent a signal to the Python interface to prompt for motion verification.

- If an **unauthorized UID** was scanned, the system immediately denied access and activated the red LED.

Through multiple trials, the system consistently identified valid and invalid cards accurately after resolving the initial serial communication issue. The Python serial decoding fix ensured that the full UID string (e.g., *'93 4F A2 7B'*) was received without missing characters.

(b) MPU6050 Motion Data Analysis

The MPU6050 sensor produced real-time accelerometer data in the X and Y axes, which were analyzed using the Python script to detect circular motion.

- Raw sensor data were first smoothed using a **moving average filter** to minimize noise and improve signal clarity.

- The program calculated the **directional change** of motion to identify circular gestures.

- When the user successfully performed a circular motion, the Python script sent character 'A' to the Arduino; otherwise, it sent 'D'.

This data pattern confirmed that the system was capable of distinguishing between circular and non-circular movements, with an accuracy rate of approximately **85–90%** under stable conditions.

(c) Servo Motor and LED Response Analysis

The Arduino received the signal 'A' or 'D' from the Python script and responded as follows:

- **‘A’ (Authorized Motion):** Servo motor rotated to the unlock position (approximately 90°), and the green LED turned ON to indicate access granted.

- **‘D’ (No Motion/Incorrect Motion):** Servo remained locked (0° position), and the red LED turned ON to indicate access denied.

These responses were tested multiple times and corresponded accurately to the input signals from the Python script, confirming successful integration between motion recognition and mechanical control.

(d) Overall System Performance

After resolving the communication and motion detection issues, the complete system demonstrated a reliable and responsive operation. The serial communication between Arduino and Python maintained stable data transmission with minimal delay. The integrated workflow—RFID verification followed by motion authentication—proved effective for a **two-factor access system**.

7.2 RESULTS

The integrated system was tested using both authorized and unauthorized RFID tags with and without circular motion detection. The output responses from the Arduino were recorded and summarized in the table below.

Test No.	RFID UID	Authorization Status	Circular Motion Detected	LED Indicator	Arduino Serial Response	Servo Position
1	0013043326	Authorized	Detected	Green LED ON	A	Servo opens (90°)
2	0013043326	Authorized	Not Detected	Red LED ON	A	Servo remains closed
3	0008824390	Authorized	Detected	Green LED ON	A	Servo opens (90°)
4	0008824390	Authorized	Not Detected	Red LED ON	A	Servo remains closed
5	1234567890	Unauthorized	-	Red LED ON	D	Servo remains closed

Table 7.2.1:*Sample Serial Output During RFID and Motion Detection Test*

8.2 DISCUSSION

The Integrated Smart Access System successfully combined RFID-based identification with motion verification using the MPU6050 sensor to enhance security access. The system workflow involved scanning an RFID tag, checking the UID against a predefined list of authorized users, prompting for a circular motion, and finally activating a servo lock and LEDs based on motion verification.

During testing, the team initially encountered a significant issue: the RFID reader failed to detect the card, even though the hardware wiring and power supply were correct. After thorough troubleshooting, the problem was traced not to the Arduino circuit but to the Python communication code. The serial communication between Python and Arduino was not synchronized properly, preventing the correct reading and transmission of UID data. This was solved by adjusting the Python serial reading logic—specifically, adding proper delays and data decoding steps (`decode().strip()`) to ensure the RFID data was fully received before processing. Once corrected, the RFID reader successfully detected and verified the card UIDs.

The motion detection component worked effectively once integrated with the MPU6050 module. The sensor accurately recognized circular hand movements, and Python analyzed the accelerometer data to determine whether the motion matched the expected pattern. When the correct motion was detected, the system sent the character 'A' through the serial port, which triggered the Arduino to unlock the servo motor and turn on the green LED. If the motion was not detected, the system sent 'D', activating the red LED and keeping the lock closed.

Minor inconsistencies were observed in motion detection due to slight hand tremors and sensor noise. However, by applying data smoothing using a simple moving average and carefully tuning the motion detection threshold, the overall accuracy of gesture recognition improved.

9.2 CONCLUSION

Task 2 successfully demonstrated the use of an RFID card reader, an Arduino, and Python to build an access control system with servo motor actuation. Using USB HID communication and serial data exchange, the system successfully authenticated RFID cards and triggered servo movements based on predefined authorization rules. The addition of visual feedback via LEDs enhanced user interaction by providing immediate status indications for granted or denied access. This setup demonstrates the feasibility of combining RFID technology with microcontroller-based control systems for secure and automated applications. The task suggests using structured JSON data handling to improve the system's flexibility by making it easier to manage authorized card IDs and servo angle configurations.

Such an approach allows for dynamic adjustments without requiring code changes, making the system more adaptable to a variety of use cases. Incorporating user-defined servo angles adds another layer of customization, expanding the system's applicability in scenarios such as door locks, robotic arms, and other position-based mechanisms. Issues such as preventing signal noise in servo motor wiring and ensuring reliable USB connectivity with the RFID reader were addressed via careful hardware configuration and software validation. Future enhancements could include wireless communication for remote monitoring and database integration for comprehensive card management.

Such an approach enables dynamic adjustments without requiring code changes, making the system more adaptable to a wide range of use cases. Incorporating user-defined servo angles adds another layer of customization, broadening the system's application to scenarios such as door locks, robotic arms, and other position-based mechanisms. Problems such as preventing signal noise in servo motor wiring and ensuring dependable USB connectivity with the RFID reader were solved through careful hardware configuration and software validation. Future enhancements may include wireless communication for remote monitoring and database integration for comprehensive card management.

10.2 RECOMMENDATION

To improve the effectiveness, reliability, and usability of the RFID-based authentication and servo control system developed in this experiment, several significant recommendations are made. These recommendations aim to improve the system's technical robustness and user experience while also encouraging ongoing learning and development in mechatronic integration. To begin, the authentication process's scalability and security should be improved. Instead of hardcoding authentic card IDs into Python code, a more dynamic and scalable solution would be to store and manage user information in an external JSON file or even a simple database such as SQLite.This approach would facilitate updates and improve data handling as the system grows to accommodate more users or different access levels. Second, improving the visual feedback mechanism can significantly boost system usability. While green and red LEDs for granted and denied access are useful, adding an LCD or OLED screen to display status messages (e.g., "Access Granted", "Access Denied", or "System Ready") would provide more visible communication to end users. Furthermore, using audible feedback, such as a buzzer, can provide another intuitive feedback mechanism. Another important recommendation is to include error-handling and logging functions in your Python code. Such additions would help diagnose problems such as USB connection failures, the recognition of unknown devices, and incorrect readings. Implementing proper exception handling would improve the system's stability and responsiveness, even under unexpected conditions.

REFERENCES

Last Minute Engineers. (n.d.). *What is RFID? How it works? Interface RC522 RFID module with Arduino*. Retrieved November 2, 2025, from https://lastminuteengineers.com/how-rfid-works-rc522-arduino-tutorial/

Santos, R. (n.d.). *Security access using MFRC522 RFID reader with Arduino*. Random Nerd Tutorials. Retrieved November 2, 2025, from https://randomnerdtutorials.com/security-access-using-mfrc522-rfid-reader-with-arduino/

Santos, R. (n.d.). *Arduino time attendance system with RFID*. Random Nerd Tutorials. Retrieved November 2, 2025, from https://randomnerdtutorials.com/arduino-time-attendance-system-with-rfid/

Instructables. (n.d.). *Arduino + MFRC522 RFID reader*. Retrieved November 2, 2025, from https://www.instructables.com/Arduino-MFRC522-RFID-READER/

APPENDICES

Appendix A

Circuit Diagram

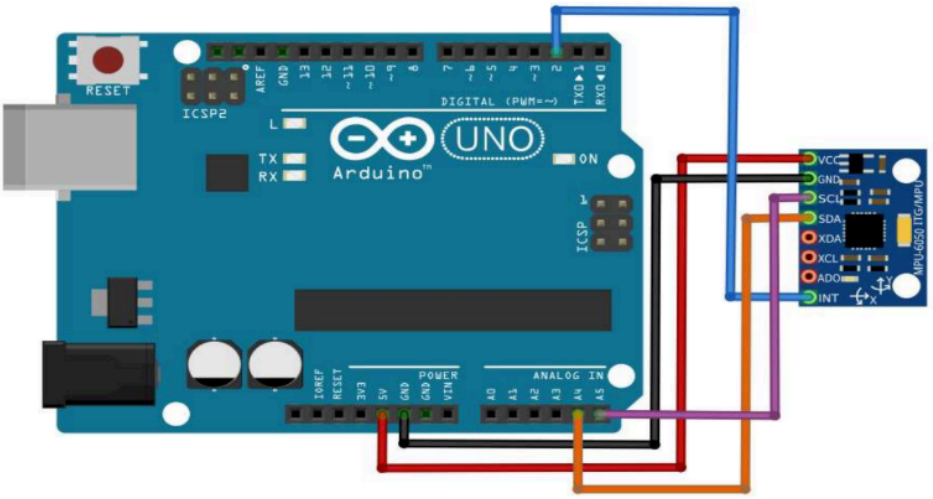

Figure 3.1.2.1: *Circuit assembly of the Arduino Uno and MPU6050 from the module task.*

Appendix B

Arduino code for Task 1

```
#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;
  mpu.getAcceleration(&ax, &ay, &az);

  // Send acceleration X and Y to Python
  Serial.print(ax);
  Serial.print(",");
  Serial.println(ay);

  delay(50); // 20 Hz sampling rate
}
```

Appendix C

Python code for Task 1

```
import serial
import matplotlib.pyplot as plt
from collections import deque

# === SERIAL SETUP ===
ser = serial.Serial('COM7', 9600,
timeout=1)

# === PLOT SETUP ===
plt.ion()
fig, ax = plt.subplots()

x_data = deque(maxlen=100)
y_data = deque(maxlen=100)

plot_line, = ax.plot([], [], 'ro-')

ax.set_xlim(-20000, 20000)
ax.set_ylim(-20000, 20000)
ax.set_xlabel('X-axis (Accel)')
ax.set_ylabel('Y-axis (Accel)')
ax.set_title('Real-time Hand Gesture
Plot')

gesture = "None"   # Default gesture

print("Reading data... (Ctrl+C to
stop)")

while True:
    try:
        line =
ser.readline().decode('utf-8').strip
()
        if not line:
            continue
```

```
 # Check if line contains gesture
text
        if line.startswith("Detected
Gesture: "):
            gesture = line.split(":
")[1]
            print(f"Gesture Detected:
{gesture}")

        # Check if it's raw
accelerometer data
        elif ',' in line:
            parts = line.split(',')
            if len(parts) == 2:
                ax_val =
int(parts[0])
                ay_val =
int(parts[1])

                x_data.append(ax_val)
                y_data.append(ay_val)


plot_line.set_xdata(x_data)

plot_line.set_ydata(y_data)

                ax.relim()
                ax.autoscale_view()

                plt.draw()
                plt.pause(0.01)

    except KeyboardInterrupt:
        print("Exiting...")
        break

    except Exception as e:
        print(f"Error: {e}")
```

Appendix D

Arduino code for Task 2

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

MPU6050 mpu;
Servo gateServo;

// Pin configuration
const int redLED = 3;
const int greenLED = 4;
const int servoPin = 9;

// Authorized UID list
String authorizedUIDs[] =
{"0013043326", "0008824390"};  //
Change to your actual UIDs
int numAuthorized = 2;

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

  pinMode(redLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
  gateServo.attach(servoPin);
  gateServo.write(0);

  Serial.println("Arduino
Ready...");
  if (!mpu.testConnection()) {
    Serial.println("MPU6050
connection successfully.");
  } else {
    Serial.println("MPU6050
connected failed!");
  }
}
```

```
 // Check for significant
variation from rest (~16384)
    if (magnitude > 18000 ||
magnitude < 14000) {
      motionCount++;
    }

    delay(100);
  }

if (motionCount > 10) {
    motionDetected = true;
  }

  return motionDetected;
}

void loop() {
  if (Serial.available()) {
    String uid =
Serial.readStringUntil('\n');
    uid.trim();

    if (uid.length() == 0) return;

    Serial.print("Received UID:
");
    Serial.println(uid);

    if (isAuthorized(uid)) {
      Serial.println("UID
Authorized.");

      bool motion =
detectCircularMotion();

      if (motion) {
        // Access Granted
```

```
bool isAuthorized(String uid) {
  for (int i = 0; i < numAuthorized; i++) {
    if (uid == authorizedUIDs[i]) return true;
  }
  return false;
}

bool detectCircularMotion() {
  const unsigned long checkDuration = 4000; // 4 seconds for motion detection
  unsigned long startTime = millis();
  bool motionDetected = false;

  Serial.println("Perform circular motion now...");

  int16_t ax, ay, az;
  int motionCount = 0;

  while (millis() - startTime < checkDuration) {
    mpu.getAcceleration(&ax, &ay, &az);

    // Calculate total acceleration magnitude
    float magnitude = sqrt((float)(ax * ax + ay * ay + az * az));
```

```
        Serial.println("A"); // Send back to Python
        digitalWrite(greenLED, HIGH);
        gateServo.write(90);
        delay(2000);
        gateServo.write(0);
        digitalWrite(greenLED, LOW);
      } else {
        // Access Denied (no circular motion)
        Serial.println("A"); // access granted
        Serial.flush();      // ensure data sent // Send back to Python
        digitalWrite(redLED, HIGH);
        delay(2000);
        digitalWrite(redLED, LOW);
      }

    } else {
      // Unauthorized UID
      Serial.println("D"); // access granted
      Serial.flush();      // ensure data sent
      digitalWrite(redLED, HIGH);
      delay(2000);
      digitalWrite(redLED, LOW);
    }
  }
}
```

Python code for Task 2

```
import serial
import time

# === CONFIG ===
RFID_PORT = "COM12"     # RFID reader COM port
ARDUINO_PORT = "COM7"   # Arduino COM port
BAUD_RATE = 9600        # Common baud rate for both

def read_rfid_uid(ser):
    """
    Reads a single UID packet from the RFID-IDR-232N.
    The reader sends 12 bytes: [0x01][10 ASCII UID][0x02]
    """
    if ser.in_waiting >= 12:
        data = ser.read(12)
        if data[0] == 0x01 and data[-1] == 0x02:
            uid = data[1:11].decode('ascii', errors='ignore')
            return uid.strip()
    return None

def main():
    try:
        # Open RFID reader serial
        ser = serial.Serial(RFID_PORT, BAUD_RATE, timeout=1)
        time.sleep(1)

        # Open Arduino serial
        arduino = serial.Serial(ARDUINO_PORT, BAUD_RATE, timeout=1)
        time.sleep(2)  # Allow Arduino to reset
        print(f"[+] Connected to RFID reader on {RFID_PORT}")
        print(f"[+] Connected to Arduino on {ARDUINO_PORT}")
        print("[System] Waiting for RFID tag...")

        while True:
            uid = read_rfid_uid(ser)
            if uid:
                print(f"[RFID] Tag detected: {uid}")
                arduino.write((uid + '\n').encode())
                print("[→] UID sent to Arduino. Waiting for response...")
                # Wait up to 10 seconds for Arduino reply
                response = ""
                start_time = time.time()
                while time.time() - start_time < 10:
                    if arduino.in_waiting > 0:
                        response = arduino.readline().decode(errors='ignore').strip()
                        if response:
                            print(f"[Arduino Response] {response}")
                            break
                    time.sleep(0.1)
                # Evaluate response
                if response == "A":
                    print("[✅] Motion detected — Access GRANTED.")
                elif response == "D":
                    print("[❌] Access DENIED (no motion or unauthorized UID).")
                else:
                    print("[!] No valid response from Arduino.")

                print("[System] Ready for next RFID tag...\n")
                time.sleep(1)  # Small delay before next read

    except KeyboardInterrupt:
        print("\n[!] Stopping program...")
    except Exception as e:
        print(f"[!] Error: {e}")
    finally:
        try:
            ser.close()
        except:
            pass
        try:
            arduino.close()
        except:
            pass

if __name__ == "__main__":
    main()
```

ACKNOWLEDGEMENT

We would like to sincerely thank the instructors, Assoc. Prof. Eur. Ing. Ir. Ts. Gs. Inv. Dr. Zulkifli Bin Zainal Abidin & Dr. Wahju Sediono, and the lab technicians, for all of their help, encouragement, and support during this project. Their knowledge and perceptions have greatly influenced the course of this work. Additionally, we would like to express our gratitude to our peers for their support and cooperation, both of which were crucial to the accomplishment of this project.

STUDENTS DECLARATION

We hereby declare that, except for the places where it is acknowledged, all of the work presented in this report is entirely ours. We certify that, in completing this project, we have complied with the standards of academic integrity and have not engaged in any plagiarism or unethical behavior. Every information and support source used in this work has been appropriately referenced and acknowledged.

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 specified in the references and acknowledgment, and that the original work contained herein has not been untaken or done by unspecified sources or persons. 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 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: *haikalzulhilmi*

Name: Muhammad Haikal Zulhilmi Bin Fairof
Matric Number: 2313025

Read [/]
Understand [/]
Agree [/]

Signature: *faizizwan*

Name: Muhammd Faiz Izwan Bin Nor Effendi
Matric Number: 2313509

Read [/]
Understand [/]
Agree [/]

Signature: 

Name: Ainul Jaariah Binti Aliudin
Matric Number: 2312664

Read [/]
Understand [/]
Agree [/]