



MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 1, 2025/2026

WEEK 4A & 4B: SERIAL INTERFACING WITH MICROCONTROLLER:

SENSORS AND ACTUATORS

SECTION 1

GROUP 20

LECTURER: DR. WAHJU SEDIONO

DATE OF EXPERIMENT: Monday, 27th October 2025

DATE OF SUBMISSION: Monday, 3rd November 2025

PREPARED BY:

NAME	MATRIC NUMBER
ALI RIDHA BIN MUHAMMAD AMINUDIN	2316137
SITI NORATHIRAH BINTI RAZALLY	2319788
NUR ALYA SAKINAH BINTI MOHD MOKHTAR	2319444

TABLE OF CONTENTS

ABSTRACT	3
INTRODUCTION	4
EXPERIMENT 4A: REAL-TIME MOON TRACKING USING MPU6050 SENSOR	5
1.0 MATERIALS AND EQUIPMENTS	5
2.0 EXPERIMENTAL SETUP	6
3.0 METHODOLOGY	7
3.1 Circuit Assembly	7
3.2 Programming Logic	7
3.3 Code Used	9
3.4 Control Algorithm	12
4.0 DATA COLLECTION	14
5.0 DATA ANALYSIS	15
6.0 RESULT	16
EXPERIMENT 4B: INTEGRATED SMART ACCESS SYSTEM	17
1.0 MATERIALS AND EQUIPMENTS	17
2.0 EXPERIMENTAL SETUP	18
3.0 METHODOLOGY	19
3.1 Circuit Assembly	19
3.2 Programming Logic	20
3.3 Code Used	22
3.4 Control Algorithm	28
4.0 DATA COLLECTION	32
5.0 DATA ANALYSIS	33
6.0 RESULT	34
DISCUSSION	35
CONCLUSION	36
RECOMMENDATIONS	37
REFERENCES	38
APPENDICES	39
Certificate of Originality and Authenticity	43

ABSTRACT

This experiment focuses on developing a motion-activated RFID access control system by integrating multiple sensors and communication interfaces with an Arduino microcontroller. The project combines an MPU6050 motion sensor, an RFID reader, a servo motor, and LED indicators to create an intelligent authentication mechanism. The MPU6050 is used to detect motion patterns while the RFID module verifies authorized users through unique identification (UID) tags. Communication between Arduino and Python enables real-time motion visualization, serial data processing, and system coordination.

In the first part, the MPU6050 is used to detect the circular motion based on the x-y coordinates. Predefined hand movements were performed, and motion data was collected and visualized using Python to observe and classify the circular motion.

In the second part, the Rfid system was programmed to detect authorized and unauthorized UID. When both RFID verification and correct motion patterns are detected, the servo motor will rotate and the green LED will light up while the red LED will light up when the wrong UID is detected.

This integration demonstrates the practical application of serial interfacing, sensor data fusion, and real-time control in mechatronic systems. The experiment successfully illustrates how sensor fusion and serial communication can enhance system security and automation in smart access control applications.

INTRODUCTION

The purpose of this experiment is to design and develop a motion-activated RFID access control system that combines the use of sensors and serial communication through an Arduino microcontroller. The project aims to create a smart security system that only grants access when a registered RFID tag is detected and a specific motion pattern is correctly performed. By completing this experiment, students gain hands-on experience in integrating sensors, actuators, and software communication between Arduino and Python.

The MPU6050 motion sensor is responsible for detecting and analyzing hand movements by using its built-in accelerometer and gyroscope to capture data along the x, y, and z axes. The RFID module, on the other hand, is used to identify authorized users through unique tag IDs. When both the correct motion and a valid RFID tag are detected, the Arduino sends a command to rotate the servo motor, simulating an unlocking mechanism, and lights up the green LED. If the RFID tag is invalid or the motion is incorrect, the system keeps the servo locked and turns on the red LED.

This experiment applies fundamental concepts of sensor fusion, serial communication, and automation in mechatronic systems. The expected outcome is that the system will successfully verify both motion and identity before granting access, showing the effectiveness of combining motion sensing and RFID technology to improve system security.

EXPERIMENT 4A: REAL-TIME MOON TRACKING USING MPU6050 SENSOR

1.0 MATERIALS AND EQUIPMENTS

1. Arduino Uno
2. Breadboard
3. MPU6050 sensor
4. Computer with Arduino IDE and Python installed
5. Connecting wires: Jumper wires or breadboard wires to establish the connections between the Arduino, MPU6050, and the power source
6. USB cable: A USB cable connects the Arduino board to your computer. This will be used for uploading the Arduino code and serial communication.

2.0 EXPERIMENTAL SETUP

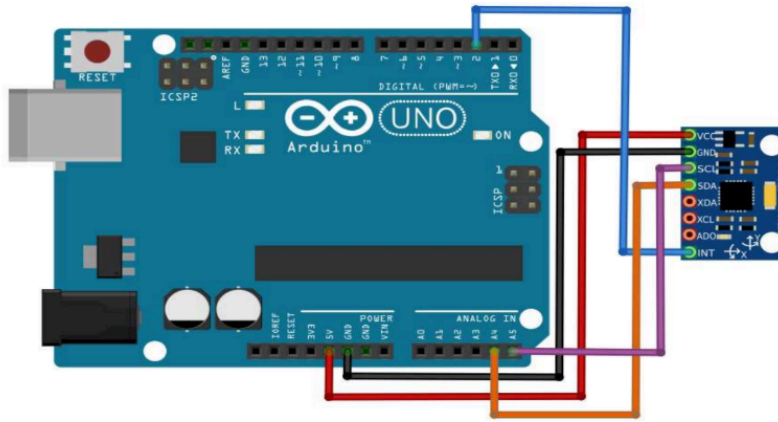


Fig. 1: Arduino-MPU6050 Connections

1. The MPU6050 sensor was connected to the Arduino using the I2C interface.
 - SDA pin → A4 (Arduino)
 - SCL pin → A5 (Arduino)
 - VCC → 5V (Arduino)
 - GND → GND (Arduino)
2. Connected the Arduino to the computer using a USB cable.
3. Uploaded the Arduino code to read data from the MPU6050 sensor.
4. Opened the Python program to receive and display the sensor data in real time.
5. Moved the MPU6050 sensor in different directions to test the motion tracking.
6. Performed a circular hand motion and observed the pattern on the graph.
7. Recorded and saved the results for analysis.

3.0 METHODOLOGY

3.1 Circuit Assembly

1. Connect an MPU6050 to the Arduino Uno
2. Ensure proper wiring
 - SDA pin → A4 (Arduino)
 - SCL pin → A5 (Arduino)
 - VCC → 5V (Arduino)
 - GND → GND (Arduino)
 - INT → D2 (Arduino)

3.2 Programming Logic

1. Setting up communication and libraries
 - Include required Arduino libraries: Wire.h and MPU6050.h for I²C communication and sensor control
 - Import Python libraries: serial for communication and matplotlib for real-time graph plotting
 - Create an MPU6050 object on the Arduino to handle sensor operations
2. Initializing systems
 - Begin serial communication at 9600 baud rate to transfer data between Arduino and Python
 - Initialize I²C communication on Arduino using Wire.begin() and configure the MPU6050 sensor
 - Test sensor connection; if the connection fails, display an error message and stop execution
 - On Python, open the serial port (e.g., COM4) to receive data from the Arduino

3. Reading and processing sensor data

- Continuously read raw accelerometer and gyroscope values (ax, ay, az, gx, gy, gz) from the MPU6050 using `getMotion6()`
- Transmit the accelerometer data (ax, ay, az) as comma-separated values to the Serial Monitor for Python to read
- In Python, continuously read and decode incoming serial data lines
- Split the received data into individual acceleration components

4. Data conversion and visualization

- Convert the raw accelerometer values to physical acceleration units (m/s²) using the formula: $\text{acceleration} = \frac{\text{raw value}}{16384.0} \times 9.81$
- Store recent data points in lists to create a smooth live graph
- Configure the plot axes and labels (e.g., Accel X vs Accel Y)

5. Displaying real-time output

- Update the scatter plot dynamically with the latest acceleration data points
- Refresh the display continuously for live visualization
- Print raw data to Serial Monitor for monitoring or debugging

6. Stabilizing and ending process

- Add a short delay (100 ms) on Arduino to ensure stable sensor readings
- Continue updating the live graph smoothly on Python until manually stopped (Ctrl + C)
- Safely close the serial connection and stop the program once data collection ends

3.3 Code Used

Arduino IDE code

```
#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() {
  int 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.println(az);
  delay(100);
}
```

Python code

```
rfid.py ×
1  import serial
2  import matplotlib
3  matplotlib.use('TkAgg')
4  import matplotlib.pyplot as plt
5
6  # === SERIAL SETUP ===
7  ser = serial.Serial(port='COM4', baudrate=9600, timeout=1)
8
9  # === PLOT SETUP ===
10 plt.ion()
11 fig, ax = plt.subplots()
12 ax.set_xlim(-20, 20) # in m/s2
13 ax.set_ylim(-20, 20)
14 ax.set_xlabel('Accel X (m/s2)')
15 ax.set_ylabel('Accel Y (m/s2)')
16 ax.set_title('MPU6050 Live Acceleration Data (X vs Y)')
17 scat = ax.scatter([], [], color='blue')
18
19 x_vals, z_vals = [], []
20
21 print("Reading acceleration data... (press Ctrl+C to stop)")
22
23 try:
24     while True:
25         line = ser.readline().decode(errors='ignore').strip()
26         if not line:
27             continue
28
```

```

28
29     try:
30         ax_raw, ay_raw, az_raw = map(float, line.split(','))
31     except ValueError:
32         continue
33
34     # Convert to m/s2 (assuming ±2g range)
35     ax_accel = (ax_raw / 16384.0) * 9.81
36     az_accel = (az_raw / 16384.0) * 9.81
37
38     x_vals.append(ax_accel)
39     z_vals.append(az_accel)
40
41     if len(x_vals) > 100:
42         x_vals.pop(0)
43         z_vals.pop(0)
44
45     scat.set_offsets(list(zip(x_vals, z_vals)))
46     plt.draw()
47     plt.pause(0.001)
48
49 except KeyboardInterrupt:
50     print("Stopped by user.")
51 finally:
52     ser.close()
53     plt.ioff()
54     plt.show()

```

3.4 Control Algorithm

1. System initialization

- Start serial communication between Arduino and computer at 9600 baud rate
- Initialize I2C communication using `Wire.begin()` to enable connection with the MPU6050 sensor
- Configure and initialize the MPU6050 module
- Verify the sensor connection; if unsuccessful, display an error message and halt the system
- On the Python side, establish serial communication by opening the corresponding COM port at the same baud rate

2. Sensor data acquisition (Arduino)

- Continuously read raw accelerometer (ax, ay, az) and gyroscope (gx, gy, gz) data from the MPU6050 using the `getMotion6()` function
- Format the accelerometer readings into a comma-separated string
- Send the formatted data through the serial port to the Python program
- Introduce a 100 ms delay to maintain stable and consistent data transmission

3. Data reception and processing (Python)

- Continuously monitor the serial port for incoming data from the Arduino
- Read and decode the serial input line by line
- Validate the data; if the line is empty or corrupted, skip and continue reading
- Split the comma-separated values into respective acceleration components (ax_raw, ay_raw, az_raw)

- Convert the raw sensor readings into physical acceleration values (m/s²) using the relation:
$$\text{acceleration} = \frac{\text{raw value}}{16384.0} \times 9.81$$

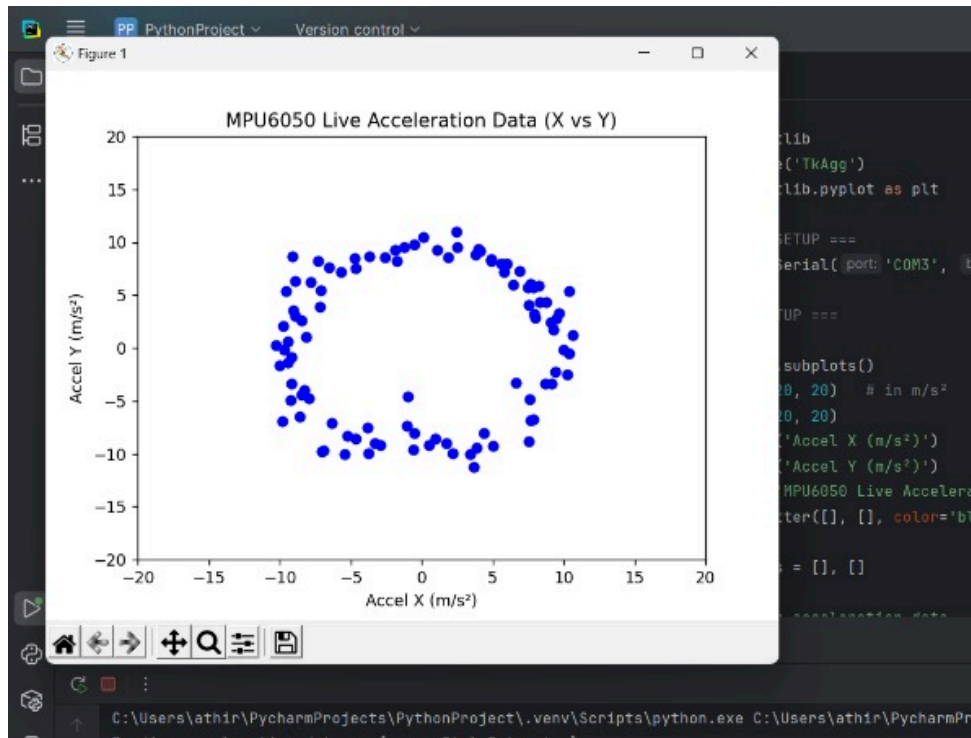
4. Data visualization control (Python)

- Initialize a real-time scatter plot using matplotlib to visualize X vs Y acceleration
- Continuously update the plot with the latest converted acceleration values
- Maintain only the most recent 100 data points to ensure smooth visual transitions
- Refresh the graph dynamically to display motion patterns in real time

5. System termination

- Continue reading and plotting data until a stop command (Ctrl + C) is received
- Upon termination, close the serial connection properly
- End the program and display the final graph for review

4.0 DATA COLLECTION



5.0 DATA ANALYSIS

In this experiment, the MPU6050 sensor was used to track motion in real time and display the movement on a computer. The sensor gave continuous readings from the x, y, and z axes, which changed depending on how the sensor was moved. When the sensor was not moving, the readings stayed almost the same, showing that it was stable. However, when the sensor was moved, especially in a circular motion, the numbers for the x and y axes changed quite noticeably. This showed that the sensor could detect direction and movement accurately.

The data collected from the sensor was sent to the computer through the Arduino using a serial connection. A Python program was used to show the readings live on a graph. When the sensor was moved in a circular motion, the graph showed a round pattern, almost like a loop. This proved that the system could track the motion correctly and visualize it in real time.

From what was observed, the results showed that the MPU6050 sensor worked well in detecting hand movements. The graph closely matched the way the sensor was moved, which means the readings were consistent and reliable. Overall, the experiment successfully showed how the MPU6050 can be used for motion tracking. This was an important step because the same motion detection was later used in Experiment 4B together with the RFID system for access control.

6.0 RESULT

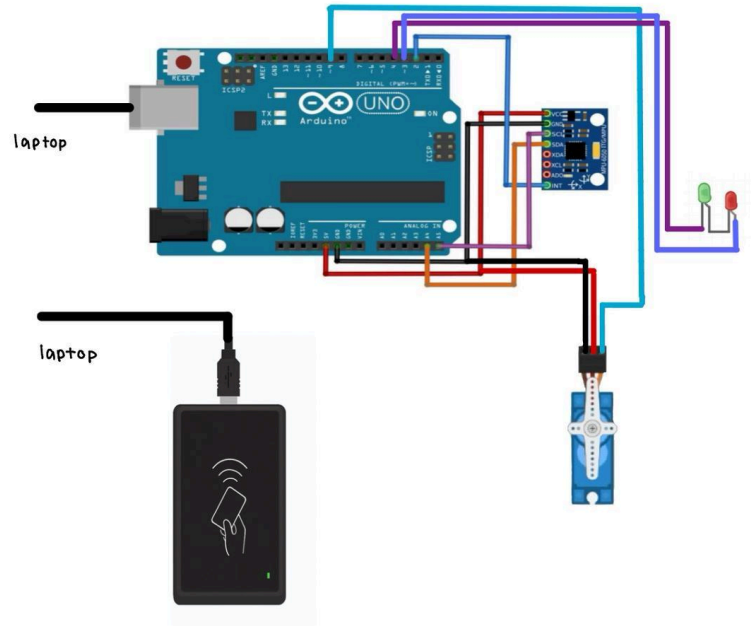
1. MPU6050 Sensor Performance: The system successfully acquired accelerometer and gyroscope data, enabling real-time data processing.
2. Gesture Detection: The Arduino successfully recognized basic gestures by comparing the acceleration values (a_x , a_y) to predefined thresholds. When a gesture was performed, the system accurately classified the gesture and transmitted the gesture data to the PC via the serial port.

EXPERIMENT 4B: INTEGRATED SMART ACCESS SYSTEM

1.0 MATERIALS AND EQUIPMENTS

1. Arduino Uno
2. RFID card reader with USB connectivity
3. RFID tags or cards that can be used for authentication
4. Servo Motor: A standard servo motor to control the angle
5. Jumper wires
6. Breadboard
7. LED red and green colors
8. USB cables to connect the Arduino board and the RFID reader to the computer
9. Computer with Arduino IDE and Python installed

2.0 EXPERIMENTAL SETUP



1. Connected the RFID reader directly to the laptop.
2. Connected the servo motor signal pin to D9 on the Arduino.
3. Connected the green LED to D4 and the red LED to D3, each with a suitable resistor.
4. Reused the MPU6050 connection from Experiment 4A.
5. Connected the Arduino to the computer using a USB cable.
6. Uploaded the Arduino code to control the servo motor and LEDs based on serial commands ('A' for access granted, 'D' for denied).
7. Ran the Python program that reads RFID tags and checks for authorized IDs.
8. Performed the required circular motion after tapping the RFID card to verify gesture detection.
9. Observed the system response:
 - Green LED and servo unlock for valid card + correct motion.
 - Red LED for invalid card or incorrect motion.
10. Recorded and saved the results for analysis.

3.0 METHODOLOGY

3.1 Circuit Assembly

1. MPU6050
 - VCC → 5V
 - GND → GND
 - SDA → A4
 - SCL → A5
2. Servo
 - Signal → Pin 9
 - VCC → 5V (use external 5V if servo is strong)
 - GND → GND (must share with Arduino)
3. Green LED
 - Anode → Pin 4 → 220Ω → LED → Cathode → GND
4. Red LED
 - Anode → Pin 3 → 220Ω → LED → Cathode → GND
5. PC connections
 - Arduino → PC via USB (serial at 9600)
 - RFID reader → PC via USB (acts like keyboard input)

3.2 Programming Logic

1. System Initialization

- Import necessary libraries: `Wire.h`, `MPU6050.h`, `Servo.h` for Arduino and `serial`, `keyboard`, `time` for Python.
- Define hardware pins for LEDs and servo motor.
- Initialize serial communication at 9600 baud rate for Arduino–Python communication.
- Setup IMU sensor (MPU6050) and test its connection.
- Set default states: LEDs off, servo locked at 90° position.

2. RFID Verification (Python Side)

- Python waits for RFID card input from a keyboard-based RFID reader.
- Compares scanned RFID data with the list of authorized card IDs.
- If the card is authorized, send signal 'A' to Arduino.
- If the card is unauthorized, send signal 'D' to Arduino.
- Provides feedback in the console (“Card verified” / “Card denied”).

3. Arduino Command Handling

- Arduino continuously checks for incoming serial data from Python.
- On receiving 'A':
 - i. Marks card as verified (`cardVerified = true`).
 - ii. Prints confirmation and begins motion detection sequence.
- On receiving 'D':
 - iii. Activates red LED for 2 seconds (access denied).
 - iv. Prints message and resets state.

4. Circular Motion Detection (IMU Sensor)

- If RFID is verified, Arduino starts detecting circular motion using MPU6050 accelerometer data.
- Reads acceleration values along x and z axes.
- Detects motion pattern through quadrant checkpoints (right, left, up, down).

- Once all 4 checkpoints are completed within a timeout (5s), circular motion is confirmed.

5. Servo and LED Control

- On successful circular motion:
 - Green LED turns ON.
 - The servo motor rotates to 180° (unlock).
 - Holds for 1 second, then returns to 90° (lock).
 - Turns LED OFF and resets verification state.
- If motion times out, the red LED activates for 2 seconds to indicate failure.

6. System Feedback and Delay

- Real-time progress messages are printed to the Serial Monitor:
 - Checkpoints count
 - Timeout or motion success status
- Short delay (100ms) used to stabilize readings and prevent command flooding.

3.3 Code Used

Arduino IDE code

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

MPU6050 imu;
Servo servoMotor;

// === Pins ===
const int LED_GREEN = 4;
const int LED_RED = 3;
const int SERVO_PIN = 9;

// === State Variables ===
bool cardVerified = false;
unsigned long motionStartTime = 0;
unsigned long motionTimeout = 5000; // 5 seconds max to complete motion

// === Motion Detection Settings ===
float xStart = 0, zStart = 0;
bool motionStarted = false;
int checkpoints = 0;

// === Setup ===
void setup() {
    Serial.begin(9600);
    Wire.begin();
    imu.initialize();

    pinMode(LED_GREEN, OUTPUT);
    pinMode(LED_RED, OUTPUT);
    servoMotor.attach(SERVO_PIN);

    // Default: both LEDs OFF, servo locked
    digitalWrite(LED_GREEN, LOW);
    digitalWrite(LED_RED, LOW);
    servoMotor.write(90);
}
```

```

if (!imu.testConnection()) {
    Serial.println("❌ MPU6050 not connected!");
    while (1);
}

Serial.println("System ready. Waiting for Python command...");
}

// === Loop ===
void loop() {
    // --- ① Wait for data from Python ---
    if (Serial.available() > 0) {
        char command = Serial.read();

        if (command == 'A') {
            cardVerified = true;
            Serial.println("✅ RFID verified. Start motion detection...");
        }
        else if (command == 'D') {
            cardVerified = false;
            Serial.println("❌ RFID denied. Red LED ON 2s...");
            digitalWrite(LED_RED, HIGH);
            delay(2000);
            digitalWrite(LED_RED, LOW);
        }
    }

    // --- ② Motion detection if card is valid ---
    if (cardVerified) {
        if (detectCircularMotion()) {
            Serial.println("✅ Circular motion verified!");
            digitalWrite(LED_GREEN, HIGH);
            digitalWrite(LED_RED, LOW);
            servoMotor.write(180); // unlock
            delay(1000);
            servoMotor.write(90); // lock again
            delay(1000);
            digitalWrite(LED_GREEN, LOW);
            cardVerified = false; // reset for next scan
        }
    }
}

```

```

    }
}

delay(100);
}

// === ③ Circular motion detection ===
bool detectCircularMotion() {
    int16_t ax, ay, az;
    imu.getAcceleration(&ax, &ay, &az);
    float x = (float)ax / 16384.0;
    float z = (float)az / 16384.0;

    if (!motionStarted && (abs(x) > 0.15 || abs(z) > 0.15)) {
        motionStarted = true;
        motionStartTime = millis();
        checkpoints = 0;
        xStart = x;
        zStart = z;
    }

    if (motionStarted) {
        // Detect approximate "circle" quadrants
        if ((x > 0.4 && abs(z) < 0.2) || // right
            (x < -0.4 && abs(z) < 0.2) || // left
            (z > 0.4 && abs(x) < 0.2) || // up
            (z < -0.4 && abs(x) < 0.2)) { // down
            checkpoints++;
            Serial.print("Checkpoint "); Serial.println(checkpoints);
            delay(150);
        }

        // Completed 4 directions = circle done
        if (checkpoints >= 4) {
            motionStarted = false;
            checkpoints = 0;
            return true;
        }

        // Timeout if taking too long
    }
}

```

```
if (millis() - motionStartTime > motionTimeout) {  
  motionStarted = false;  
  checkpoints = 0;  
  Serial.println("⌚ Motion timeout!");  
  digitalWrite(LED_RED, HIGH);  
  delay(2000);  
  digitalWrite(LED_RED, LOW);  
  return false;  
}  
}  
  
return false;  
}
```

Python code

```
python code_task 2.py ×
1  import serial
2  import time
3  import keyboard # pip install keyboard
4
5  # === Serial Settings ===
6  SERIAL_PORT = 'COM4'
7  BAUD_RATE = 9600
8
9  # === Authorized RFID IDs ===
10 authorized_cards = ["0008089233"]
11
12
13 def read_rfid():
14     """Reads RFID card from keyboard-type reader."""
15     print("Please tap your RFID card...")
16     rfid_data = ""
17     while True:
18         event = keyboard.read_event()
19         if event.event_type == keyboard.KEY_DOWN:
20             key = event.name
21             if key == 'enter':
22                 print(f"RFID scanned: {rfid_data}")
23                 return rfid_data.strip()
24             elif len(key) == 1:
25                 rfid_data += key
26
27
28 def main():
29     try:
30         arduino = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
31         time.sleep(2)
32         print("Connected to Arduino.")
33
```

```

34         while True:
35             card = read_rfid()
36             if card in authorized_cards:
37                 print("✅ Card verified. Proceed with circular motion.")
38                 arduino.write(b'A') # authorized
39             else:
40                 print("❌ Card denied.")
41                 arduino.write(b'D') # denied
42
43             time.sleep(0.5)
44
45         except KeyboardInterrupt:
46             print("Program terminated.")
47         finally:
48             if 'arduino' in locals() and arduino.is_open:
49                 arduino.close()
50
51
52 if __name__ == "__main__":
53     main()

```



3.4 Control Algorithm

1. System Initialization

- The system begins by initializing all required components on both Arduino and Python sides.
- Arduino:
 - Imports libraries `Wire.h`, `MPU6050.h`, and `Servo.h` to enable IMU and servo functions.
 - Configures hardware pins for the green LED, red LED, and servo motor.
 - Establishes serial communication at 9600 baud rate with the Python program.
 - Initializes the MPU6050 sensor and verifies its connection.
 - Sets both LEDs to OFF and positions the servo motor at 90° (locked).
- Python:
 - Imports `serial`, `time`, and `keyboard` libraries.
 - Establishes serial connection with the Arduino using the correct COM port and baud rate.
 - Prepares a list of authorized RFID card IDs for verification.

2. RFID Authentication Process

- The Python program continuously waits for an RFID card to be scanned using a keyboard-type RFID reader.
- Once a card is detected, its unique ID is read and compared against the stored authorized list.
- If the card is authorized:

- Python sends command 'A' through the serial port to Arduino.
- Displays the message “ Card verified. Proceed with circular motion.”
- If the card is unauthorized:
 - Python sends command 'D' through the serial port.
 - Displays “ Card denied.” on the console.
- The Arduino continuously listens for serial data and responds accordingly.

3. Command Response and Verification

- Upon receiving command 'A':
 - Arduino sets `cardVerified = true`, allowing the motion detection process to begin.
 - Displays confirmation on the Serial Monitor.
- Upon receiving command 'D':
 - Arduino turns ON the red LED for 2 seconds to indicate access denial.
 - Turns OFF the LED and resets `cardVerified` to false.
 - Displays “RFID denied” message on Serial Monitor.

4. Motion Detection Control

- If the card is verified, the Arduino begins circular motion detection using the MPU6050 accelerometer.
- The system continuously reads acceleration values along the x-axis and z-axis.
- Motion starts when the detected acceleration exceeds a predefined threshold (± 0.15 g).
- The program tracks checkpoints representing four motion directions: right, left, up, and down.

- Each valid direction is counted as a checkpoint toward completing one circular motion.
- If four checkpoints are detected within the time limit (5 seconds):
 - The system recognizes a successful circular motion.
- If motion exceeds the time limit without completion:
 - Timeout is triggered, and the motion process resets.

5. Actuation and Output Control

- Successful Motion:
 - Arduino turns ON the green LED to indicate successful motion verification.
 - The servo motor rotates to 180° to simulate unlocking.
 - After a short delay, the servo returns to 90° (locked position).
 - Green LED turns OFF, and `cardVerified` is reset for the next cycle.
- Unsuccessful Motion (Timeout):
 - The red LED activates for 2 seconds to indicate failure or timeout.
 - Motion variables and checkpoints are reset to prepare for the next attempt.

6. Feedback and Delay Handling

- Throughout the process, Arduino sends real-time status messages to the Serial Monitor:
 - RFID verification results
 - Motion checkpoint counts
 - Success or timeout notifications

- A short delay (100 ms) is introduced in the main loop to stabilize communication, prevent rapid signal fluctuations, and ensure smooth transitions between control stages.
- The Python program also continuously prints verification updates for user monitoring.

4.0 DATA COLLECTION

UID	Accessible	Green LED	Red LED	Servo
0008089233	Granted	Turn On	Turn Off	Rotate 90° and turn back to 0°
0013084287	Denide	Turn Off	Turn On	Remains 0°

5.0 DATA ANALYSIS

In Experiment 4B, the RFID module and the MPU6050 motion sensor were combined to create a smart access control system. The RFID reader was used to detect and identify the user through a unique tag ID, while the MPU6050 sensor verified the motion pattern, which in this case was a circular hand movement. Both components worked together with the Arduino and Python programs to decide whether access should be granted or denied.

During the experiment, several RFID tags were tested. When a valid tag was tapped on the reader, the system requested a circular hand motion. If the motion was performed correctly, the Python program sent a signal to the Arduino, which unlocked the servo motor and turned on the green LED. If an unregistered tag was used or the motion was incorrect, the servo motor stayed locked and the red LED turned on.

From the observations, the system successfully recognized the authorized RFID tag and responded correctly to the motion gesture. The LEDs and servo motor gave clear feedback that matched the program's conditions. This proved that both the RFID and motion detection systems were functioning as expected.

Overall, the experiment showed that combining RFID verification with motion detection can create a more secure access system. The data confirmed that the integrated setup can accurately identify users and verify gestures before granting access, demonstrating how sensor fusion and serial communication can be applied in real-world security systems.

6.0 RESULT

1. **UID Detection:** The RFID reader successfully detected and read the UID of the RFID tags. The system correctly transmitted the UID to a Python program for validation.
2. **Access Validation:** The Python program correctly compared the read UID against a pre-stored list in Python list format and successfully determined whether the UID was valid or invalid. Valid UIDs triggered visual feedback via LEDs and servo motor actuation, simulating a real-world access control mechanism.
3. **System Response:** When a valid RFID tag was detected, the system provided immediate visual feedback, such as rotating the servo motor, confirming that the access control mechanism was functioning as intended.

DISCUSSION

This experiment explored two standalone embedded systems: a gesture recognition system using the MPU6050 sensor and an RFID-based access control system. Both systems were designed to demonstrate real-time sensor data handling, classification, and actuator control, aligning with modern applications in human-computer interaction and secure access management.

LIMITATIONS & SOURCES OF ERROR:

Experiment 4A: Real-Time Motion Tracking Using MPU6050 Sensor

1. Fixed thresholds may not generalize well across different users or movement intensities.
2. Sensor readings were susceptible to noise, drift, and external vibrations.
3. Gesture overlap or slight movements could lead to misclassification.
4. The 100ms loop delay, while smoothing data, introduced minor latency.
5. No filtering or sensor fusion was applied, which could have improved gesture stability and accuracy.

Experiment 4B: Integrated Smart Access System

1. RFID readers can sometimes fail to read a tag if the orientation or distance isn't optimal.
2. Manual UID registration can become inefficient as the user database scales.
3. No encryption was implemented for data transmission, making it less secure in a real-world application.
4. Serial communication speed must be managed carefully to avoid delays or data corruption, especially during rapid scanning.

CONCLUSION

In Experiment 4A, the experiment successfully demonstrated the use of the MPU6050 sensor to perform real-time motion tracking. Accelerometer data were transmitted from the Arduino to Python via serial communication and visualized dynamically using the `matplotlib` library. The system effectively detected circular motion patterns, validating the implementation of basic motion recognition techniques and achieving the intended learning objectives.

In Experiment 4B, the RFID, MPU6050, and servo motor were successfully integrated to form a smart access control system. The Python program accurately compared the scanned UID against a pre-stored list in Python list format and determined whether the UID was valid or invalid. When a valid RFID card was detected and the correct circular motion was performed, the servo motor was activated to unlock the system while the green LED indicated successful access. Conversely, invalid UIDs or incorrect motion patterns triggered the red LED to indicate access denial. Overall, the integrated system demonstrated effective communication between Python and Arduino, successfully combining RFID authentication and motion-based verification for secure access control.

RECOMMENDATIONS

For future versions of this project, a few improvements can be made to make the system work more smoothly and accurately. One of the main things that can be improved is the power connection for the servo motor. Instead of using power directly from the Arduino, it would be better to use an external 5V power supply so that the servo moves more steadily without any shaking. It's also a good idea to double-check the voltage for the RFID module since some of them only work at 3.3V. Using the right voltage or a level shifter can help protect the component from damage. Keeping the wiring short and neat on the breadboard also helps to reduce noise and avoid unstable readings from the MPU6050 sensor.

Calibrating the MPU6050 sensor before starting the test is also very important. This step ensures that the readings are more stable and accurate. Adding a simple data filter in the code, such as a moving average, could make the motion detection smoother and reduce sudden spikes in the data. For even better accuracy, both the accelerometer and gyroscope readings can be combined to make the circular motion detection more precise.

From this experiment, several useful lessons were learned. One of them is that serial communication between Arduino and Python needs to be handled carefully. The serial monitor must be closed before running the Python program to prevent connection errors. Another thing learned was how small details like loose wires or long jumper cables can cause unstable readings. It's always best to keep everything properly connected and to test each part separately before combining them.

Overall, this project was a great learning experience. It showed how important it is to be patient when working with sensors and communication systems. Taking time to calibrate, organize connections, and test step by step made a big difference in getting accurate results. Future students working on similar projects can benefit from these lessons by setting up everything carefully and checking each component before integration. This will help them avoid common mistakes and achieve better and more consistent results.

REFERENCES

Last Minute Engineers. (2018, July 30). *What is RFID? How It Works? Interface RC522 RFID Module with Arduino*. Last Minute Engineers; Last Minute Engineers.
<https://lastminuteengineers.com/how-rfid-works-rc522-arduino-tutorial/>

Santos, R. (2016, March 23). *MFRC522 RFID Reader with Arduino Tutorial | Random Nerd Tutorials*. Random Nerd Tutorials.
<https://randomnerdtutorials.com/security-access-using-mfrc522-rfid-reader-with-arduino/>

RFID-Based Attendance System Using Arduino. (2025). Arduino Project Hub.
<https://projecthub.arduino.cc/rinme/rfid-based-attendance-system-using-arduino-9e45eb>

APPENDICES

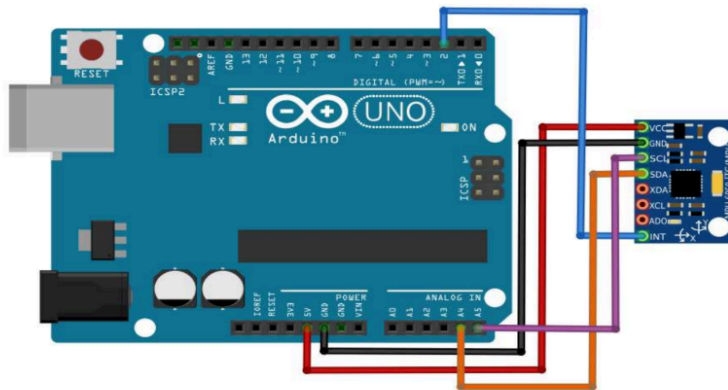
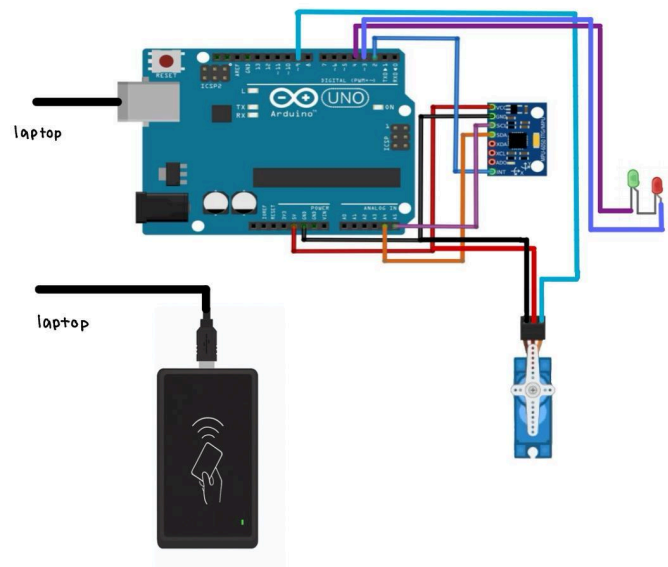
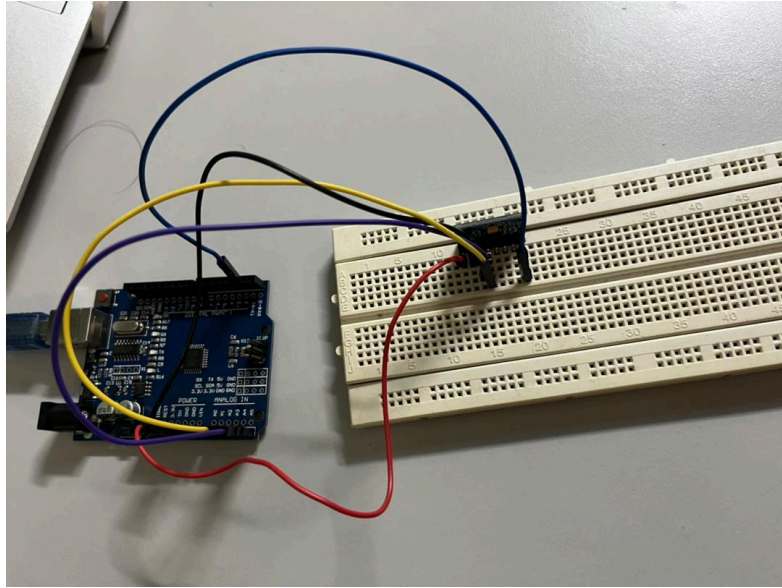


Fig. 1: Arduino-MPU6050 Connections

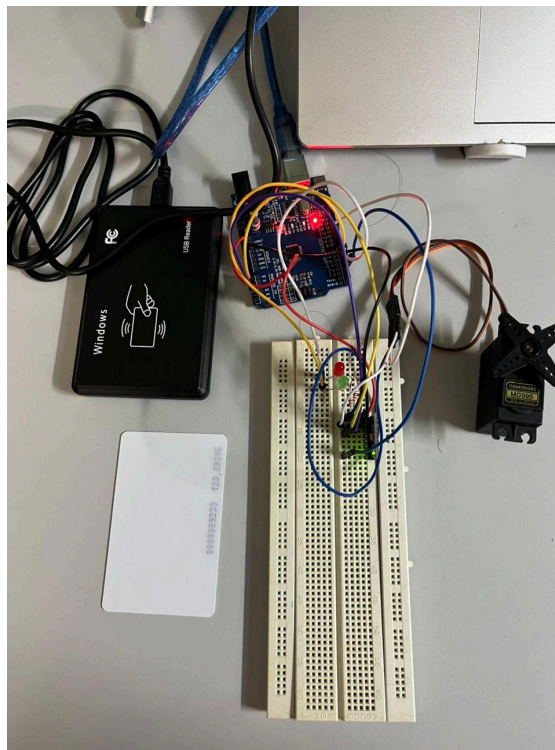
Wiring diagram for Experiment 4A



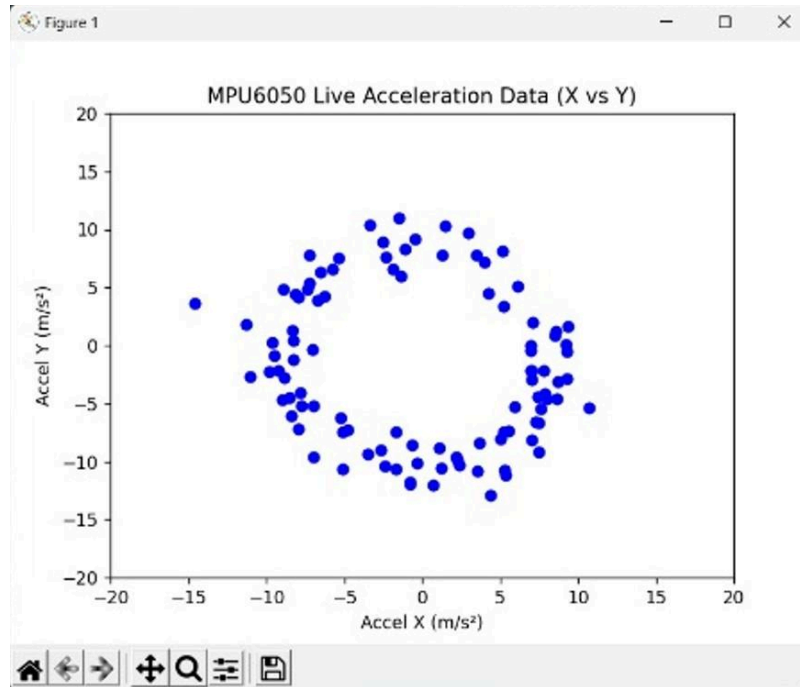
Wiring diagram for Experiment 4B



Experiment 4A



Experiment 4B




Circular motion pattern

ACKNOWLEDGEMENTS

Special thanks to Dr. Wahju Sediono for their guidance and support during this experiment. We would also like to extend our gratitude to our instructors, peers, and the laboratory staff for their valuable insights and assistance throughout the experiment. Their feedback and encouragement have greatly contributed to the success of this project. Finally, we appreciate the resources and learning materials provided, which enabled us to understand and implement serial communication effectively.

Certificate of Originality and Authenticity

We hereby certify that we are responsible for the work presented in this report. The content is our original work, except where proper references and acknowledgements are made. We confirm that no part of this report has been completed by anyone not listed as a contributor. We also certify that this report is the result of group collaboration and not the effort of a single individual. The level of contribution by each member is stated in this certificate. Furthermore, we have read and understood the entire report, and we agree that no further revisions are required. We collectively approve this final report for submission and confirm that it has been reviewed and verified by all group members.

Signature: 
Name: ALI RIDHA BIN MUHAMMAD AMINUDIN
Matric Number: 2316137

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

Signature: 
Name: SITI NORATHIRAH BINTI RAZALLY
Matric Number: 2319788

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

Signature: 
Name: NUR ALYA SAKINAH BINTI MOHD MOKHTAR
Matric Number: 2319444

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