

MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 1 2025/2026

WEEK 4: SERIAL INTERFACING WITH MICROCONTROLLER: SENSORS AND
ACTUATORS

DATE OF EXPERIMENT: 29 OCTOBER 2025

DATE OF SUBMISSION: 5 NOVEMBER 2025

SECTION 1

GROUP 13

LECTURER: ZULKIFLI BIN ZAINAL ABIDIN

NO.	NAME	MATRIC
1	HARIZ IRFAN BIN MOHD ROZHAN	2318583
2	ABDULLAH HASAN BIN SIDEK	2318817
3	TAN YONG JIA	2319155
4	NUR QISTINA BINTI MOHD FAIZAL	2319512

ABSTRACT

This experiment focuses on designing and implementing a motion-activated RFID access control system using an Arduino microcontroller, MPU6050 motion sensor, RFID module, and servo motor. The objective is to integrate sensor data acquisition, serial communication, and actuator control to create a smart authentication mechanism. In Part 1, the MPU6050 sensor was interfaced with the Arduino to capture real-time accelerometer data, which was processed and visualized using Python to detect motion patterns, specifically a circular hand gesture. In Part 2, the RFID module was used to identify authorized users, while the servo motor and LEDs provided access control feedback. The system granted access only when both RFID verification and the correct motion gesture were detected. This experiment demonstrates the integration of hardware and software for intelligent mechatronic systems, emphasizing sensor fusion, serial data communication, and Python–Arduino interfacing for real-time decision-making applications such as secure entry systems.

TABLE OF CONTENTS

ABSTRACT	2
TABLE OF CONTENTS	2
1.0 INTRODUCTION	3
TASK 1	3
2.0 MATERIALS AND EQUIPMENTS	3
3.0 EXPERIMENTAL SETUP	4
4.0 METHODOLOGY	4
5.0 DATA COLLECTION	9
6.0 DATA ANALYSIS	9
7.0 RESULTS	10
8.0 DISCUSSION	10
9.0 CONCLUSION	11
10.0 RECOMMENDATIONS	12
11.0 REFERENCES	12
12.0 APPENDICES	13
13.0 ACKNOWLEDGEMENTS	14
14.0 STUDENTS DECLARATION	15

1.0 INTRODUCTION

In today's world, security and automation systems are becoming increasingly intelligent through the integration of sensors, microcontrollers, and software-based data processing. Traditional access control systems, such as RFID authentication, offer convenience but can still be enhanced with additional verification methods. This experiment introduces a motion-activated RFID system that combines RFID identification with motion gesture recognition using an MPU6050 sensor and Arduino microcontroller. The system is designed to grant access only when a valid RFID tag is detected and a specific motion pattern, such as a circular hand gesture, is performed.

The experiment is divided into two main parts. In Part 1, the focus is on reading real-time accelerometer and gyroscope data from the MPU6050 sensor and visualizing the motion path dynamically in Python using serial communication. This establishes the foundation for detecting specific motion gestures. In Part 2, the system integrates the RFID reader, servo motor, and LEDs with the motion detection logic to form a complete smart access mechanism. Authorized RFID tags trigger a motion verification step, and only when the correct gesture is recognized does the servo unlock while a green LED indicates access granted.

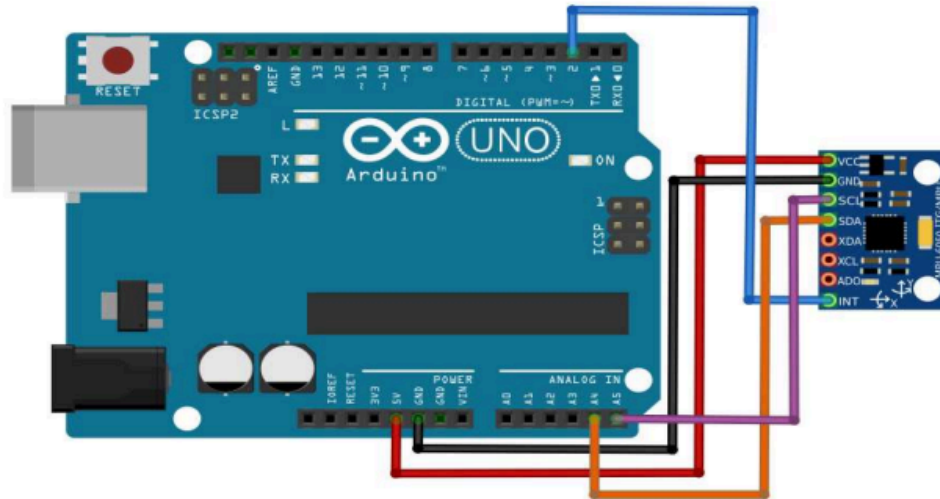
Through this experiment, students gain hands-on experience in serial interfacing, sensor integration, and actuator control, which are essential elements of mechatronic system design. It also highlights how hardware and software collaboration can be applied to create efficient and secure real-world automation systems.

TASK 1

2.0 MATERIALS AND EQUIPMENTS

- Arduino board
- MPU6050 sensor
- Jumper wires/breadboard
- USB cable
- Computer with Arduino IDE and Python installed

3.0 EXPERIMENTAL SETUP



4.0 METHODOLOGY

1. The Arduino Uno was linked to the MPU6050 sensor in line with the circuit configuration.
2. In order to continuously transmit acceleration data from the sensor via the serial port, the Arduino code was uploaded.
3. The acceleration data was received and shown in real time via a Python program that made use of the PySerial and Matplotlib libraries.
4. The COM4 port was used to establish serial communication at a baud rate of 9600 bps between the Arduino and computer.
5. The software dynamically plotted the X, Y, and Z acceleration values from the sensor on a scatter graph after continuously reading them.
6. The code produced a real-time graph that displayed the correlation between the X-axis and Z-axis acceleration.
7. After manually stopping the program, the finished graph was shown for viewing..

4.1 Circuit Assembly

- The Arduino Uno was linked to the MPU6050 sensor via the I²C communication interface.

- The 5V and GND pins of the Arduino were linked to the VCC and GND pins of the sensor, respectively.
- The SDA pin of the MPU6050 was linked to the A4 pin of the Arduino, while the SCL pin was linked to A5.
- A USB cable was then used to connect the Arduino Uno to the computer, enabling data transfer and power supply.
- With this configuration, the Arduino could read acceleration data from the MPU6050 and transmit it to the computer so that the Python program could visualise it in real time.

4.2 Programming Logic

- Using the I²C communication interface, the Arduino code continuously retrieves raw acceleration data from the MPU6050 sensor.
- The X, Y, and Z axes' acceleration values are included in these readings.
- The `Serial.println()` function is then used to send the data in a separated by commas format to the computer via serial communication.
- This data is sent to a Python program via the serial port via the PySerial library.
- The acceleration values are extracted from the incoming serial data, decoded, and updated in real time by the Python script.
- A live scatter graph illustrating the relationship between the X-axis and Z-axis acceleration is displayed by dynamically plotting the acceleration values using the Matplotlib library.
- Until the user manually stops it, the program keeps updating the graph.

4.3 Code used :

Arduino 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:

```

22
23     try:
24         while True:
25             line = ser.readline().decode(errors='ignore').strip()
26             if not line:
27                 continue
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)

```

```

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, right=20) # in m/s2
13 ax.set_ylim(-20, top=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(x=[], y=[], 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
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49 except KeyboardInterrupt:
50     print("Stopped by user.")
51 finally:
52     ser.close()
53     plt.ioff()
54     plt.show()

```

4.4 Control Algorithm

1. Defining pins:

The Arduino is linked to the MPU6050 sensor via I²C communication pins:

- SDA → A4
- SCL → A5
- VCC → 5V

- GND → GND

Through the USB serial port, the Arduino and computer can communicate, transferring acceleration data for Python visualisation.

2. Global variables:

Several global variables are declared to store the raw sensor data:

- int ax, ay, az → store accelerometer readings for the X, Y, and Z axes.
- int gx, gy, gz → store gyroscope readings for rotational motion.

These variables are updated continuously to reflect the sensor's real-time motion data.

3. Setup function:

- To enable data transmission to the Python program, Serial.begin(9600) is used to initialise serial communication.
- To enable communication with the MPU6050, Wire.begin() is used to initiate the I²C communication.
- mpu.initialize() is used to initialise the MPU6050.
- mpu.testConnection() is used to confirm the connection.

The application stops and displays an error message that reads, "MPU6050 connection failed!" if the connection is lost.

4. Main loop:

These actions are continuously carried out by the loop() function:

I. Check the sensor data:

- The function mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz) returns six raw sensor values: gyroscope rotation and three-axis acceleration.

II. Data transmission via serial:

- The accelerometer values (ax, ay, az) are sent to the computer through the serial port.
- The data is formatted as **comma-separated values**:
ax, ay, az
making it easier for the Python program to parse.

III. Delay:

- To keep the data stream consistent and avoid data overflow, a 100 ms delay is added.

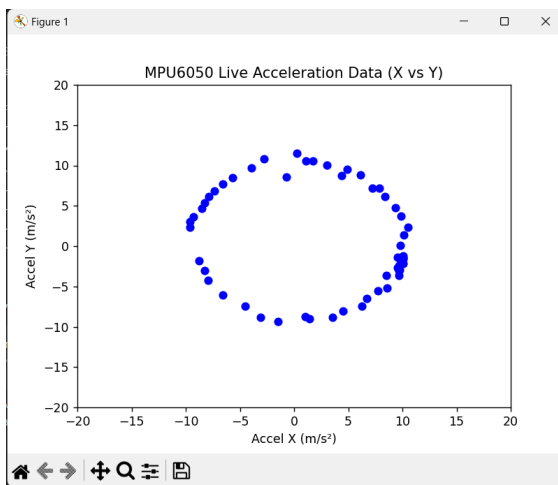
5. Python visualisation algorithm:

On the computer side, a Python program reads the serial data and plots the acceleration in real time:

- The application starts an interactive Matplotlib scatter plot and establishes a serial connection (COM4, 9600 baud).
- Incoming data lines are divided into X, Y, and Z components.
- The following conversion formula is used to convert the raw values to m/s^2 :
acceleration is equal to $(\text{raw_value} / 16384.0) * 9.81$.
(Assuming sensitivity of $\pm 2\text{g}$.)
- To create a real-time motion trace, the converted values are plotted as X vs. Z acceleration after being added to lists.
- The graph, which illustrates how the sensor's acceleration varies with movement, automatically updates at brief intervals (`plt.pause(0.001)`).

5.0 DATA COLLECTION

Observation:



6.0 DATA ANALYSIS

- Acceleration is measured along the X, Y, and Z axes by the MPU6050 sensor.
- Through serial communication, Arduino provides the computer with raw acceleration data.
- Python uses the following to convert raw values to m/s²:
$$a = (\text{raw value} / 16384.0) \times 9.81$$
- X vs. Z acceleration is shown in a real-time scatter plot using Matplotlib.
- Movement disperses data points, while a stationary sensor displays clustered points.
- The system performs responsively and reliably, updating without any issues.
- All things considered, the configuration effectively displays motion data from the MPU6050 in real time.

7.0 RESULT

The experiment effectively illustrated how to use the MPU6050 sensor to acquire data in real-time and visualise acceleration measurements. Raw accelerometer data (in the X, Y, and Z axes) was continuously sent from the Arduino to the Python interface via the serial port. The data were transformed into acceleration values (m/s²) in the Python environment, and a live scatter plot displaying X-axis vs. Z-axis acceleration was then displayed. The displayed points dynamically changed in response to tilting or moving the MPU6050 sensor, revealing the acceleration's direction and amplitude. The graph's obvious response to sensor movement demonstrated that the Arduino and Python were communicating properly. When the sensor remained still, the points clustered near the origin, whereas movement caused distinct point dispersion on the plot. This verified that the MPU6050 sensor was accurately capturing and transmitting motion data in real time.

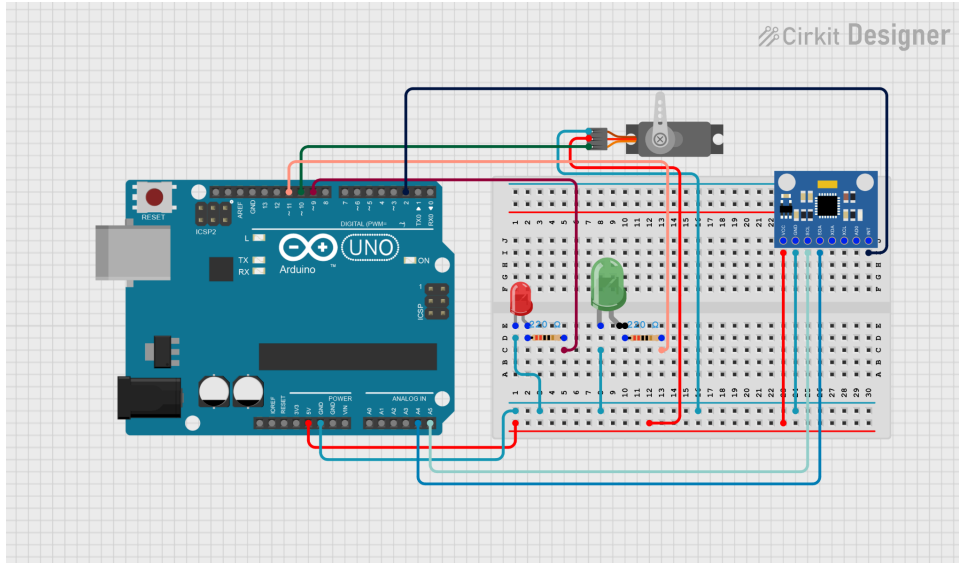
TASK 2

2.0 MATERIALS AND EQUIPMENTS

- Arduino board
- RFID reader (MFRC522) + RFID tags/cards
- Servo motor
- Red & Green LEDs + resistors

- MPU6050 sensor (from Part 1)
- Jumper wires and breadboard

3.0 EXPERIMENTAL SETUP



4.0 METHODOLOGY

1. The Arduino Uno was connected to the RFID module, MPU6050 sensor, servo motor, and LED indicators according to the circuit configuration.
2. The Arduino code was uploaded to continuously read acceleration data from the MPU6050 and wait for serial commands from the computer.
3. A Python program was created using the PySerial and NumPy libraries to manage communication, motion detection, and access control.
4. The COM3 port was used to establish serial communication between the Arduino and the computer at a baud rate of 9600 bps.
5. During operation, the user scanned an RFID card ID using the Python interface.
6. The Python script compared the scanned card ID with the authorized ID stored in the program.
7. If the card was authorized, the system collected MPU6050 acceleration data for 2 seconds to analyze motion.

8. The Python program calculated the motion score by comparing the new readings to the baseline calibration.
9. If the motion score exceeded the threshold, the Python script sent the command '1' (Access Granted) to the Arduino.
10. If no motion or an unauthorized card was detected, the command '2' (Access Denied) was sent instead.
11. When the Arduino received '1', it turned ON the green LED, moved the servo to the unlock position for 3 seconds, then returned it to the locked position.
12. When the Arduino received '2', it turned ON the red LED for 2 seconds and kept the servo locked.
13. The process repeated continuously, allowing multiple RFID scans and motion detection trials throughout the experiment.

4.1 Circuit Assembly

- The Arduino Uno was connected to the MPU6050 sensor using the I²C communication interface.
- The VCC and GND pins of the MPU6050 were connected to the 5V and GND pins of the Arduino, respectively.
- The SDA pin of the sensor was linked to A4, and the SCL pin was linked to A5 on the Arduino Uno.
- The RFID module (MFRC522) was connected to the Arduino using the SPI interface.
- The SDA, SCK, MOSI, and MISO pins of the RFID reader were connected to D10, D13, D11, and D12, respectively, while the RST pin was connected to D8.
- The module's 3.3V and GND pins were connected to the Arduino's 3.3V and GND pins.
- A servo motor was connected to digital pin 10, with its VCC connected to 5V and GND connected to the Arduino ground.
- Two indicator LEDs were added: the green LED connected to D11 through a 220 Ω resistor and the red LED connected to D9 through another resistor, both with their cathodes tied to GND.
- Finally, the Arduino Uno was connected to the computer via a USB cable, providing both power and serial communication.

- With this setup, the Arduino could receive card data from the RFID reader, motion data from the MPU6050, and control the servo and LEDs according to the Python program's commands.

4.2 Programming Logic

- Arduino uses I²C to read acceleration data (X, Y, Z) from the MPU6050 sensor.
- The data is sent to the computer via Serial communication using Serial.println().
- Python program connects to Arduino using PySerial at 9600 bps.
- The system performs baseline calibration by averaging sensor data for 3 seconds.
- The user scans an RFID card (entered in Python console).
- Python checks if the scanned UID matches the authorized ID.
- If authorized:
 - Collect MPU6050 data for 2 seconds.
 - Calculate motion score by comparing current data with baseline.
 - If motion score > threshold → send '1' (Access Granted).
 - Else → send '2' (Access Denied).
- If unauthorized:
 - Send '2' to Arduino (Access Denied).
- Arduino response:
 - '1': Green LED ON, servo unlocks (60°), waits 3s, then locks (0°).
 - '2': Red LED ON for 2s, servo stays locked.
- The process repeats continuously until the user stops the Python program.

4.3 Code used :

Arduino code:

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

MPU6050 mpu;
Servo myServo;

const int GREEN_LED = 11;
const int RED_LED = 9;
const int SERVO_PIN = 10;

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

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

  pinMode(GREEN_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  myServo.attach(SERVO_PIN);
  myServo.write(0);

  Serial.println("System Ready - Waiting for PC...");
}
```

```

void loop() {
  // --- Continuously send MPU data for Python to read ---
  int16_t ax, ay, az;
  mpu.getAcceleration(&ax, &ay, &az);
  Serial.print("AcX="); Serial.print(ax);
  Serial.print("|AcY="); Serial.print(ay);
  Serial.print("|AcZ="); Serial.println(az);
  delay(50);

  // --- Listen for Python commands ---
  if (Serial.available()) {
    char cmd = Serial.read();

    if (cmd == '1') { // Authorized + Motion detected
      Serial.println("✅ Authorized & Motion detected!");
      digitalWrite(GREEN_LED, HIGH);
      digitalWrite(RED_LED, LOW);
      myServo.write(60);
      delay(3000);
      myServo.write(0);
      digitalWrite(GREEN_LED, LOW);
    }
    else if (cmd == '2') { // Unauthorized or No motion
      Serial.println("❌ Access denied!");
      digitalWrite(GREEN_LED, LOW);
      digitalWrite(RED_LED, HIGH);
      myServo.write(0);
      delay(2000);
      digitalWrite(RED_LED, LOW);
    }
  }
}

```

```
}  
}
```

Python code:

```
import serial  
import time  
import numpy as np  
  
# --- Serial Setup ---  
ARDUINO_PORT = "COM3" # Change if needed  
BAUD_RATE = 9600  
arduino = serial.Serial(ARDUINO_PORT, BAUD_RATE, timeout=1)  
time.sleep(2)  
print(f"✅ Connected to Arduino on {ARDUINO_PORT}")  
  
# --- Authorized RFID UID ---  
AUTHORIZED_ID = "0003855091"  
  
# --- Helper: Read one MPU6050 line ---  
def read_mpu_line():  
    """Reads one sensor line in the form 'AcX=...|AcY=...|AcZ=...'"""  
    try:  
        line = arduino.readline().decode(errors='ignore').strip()  
        if not line or "AcX" not in line:  
            return None  
        parts = line.replace(" ", "").split("|")  
        ax = int(parts[0].split("=")[1])  
        ay = int(parts[1].split("=")[1])  
        return ax, ay  
    except Exception:
```

```

    return None

# --- Baseline Calibration ---
print("📏 Calibrating baseline (keep sensor still for 3s)...")
baseline_ax, baseline_ay = [], []
start_time = time.time()
while time.time() - start_time < 3:
    data = read_mpu_line()
    if data:
        ax, ay = data
        baseline_ax.append(ax)
        baseline_ay.append(ay)
    time.sleep(0.05)

if baseline_ax and baseline_ay:
    mean_ax = np.mean(baseline_ax)
    mean_ay = np.mean(baseline_ay)
    print(f"✅ Baseline set: AX={mean_ax:.2f}, AY={mean_ay:.2f}")
else:
    mean_ax = mean_ay = 0
    print("⚠️ No baseline data, defaulting to 0.")

# --- Main Loop ---
print("\nRFID + Motion System Running...\n")

while True:
    try:
        card = input("Tap card: ").strip()
        print(f"Card Scanned: {card}")

        if card == AUTHORIZED_ID:

```

```

print("✅ Authorized card detected! Move the sensor now...")

# Collect MPU data for 2 seconds
ax_samples, ay_samples = [], []
start_time = time.time()
while time.time() - start_time < 2:
    data = read_mpu_line()
    if data:
        ax, ay = data
        ax_samples.append(ax)
        ay_samples.append(ay)
    time.sleep(0.05)

if ax_samples and ay_samples:
    # --- Smoothed motion detection based on deviation from baseline ---
    ax_deltas = [abs(ax - mean_ax) for ax in ax_samples]
    ay_deltas = [abs(ay - mean_ay) for ay in ay_samples]
    motion_score = np.mean(ax_deltas + ay_deltas)

    print(f"📊 Motion Score: {motion_score:.2f}")

    # --- Motion threshold (tune as needed) ---
    if motion_score > 100:
        print("✅ Motion detected! Sending '1' to Arduino.")
        arduino.reset_input_buffer()
        arduino.write(b'1')
        arduino.flush()
    else:
        print("❌ No significant motion. Sending '2' to Arduino.")
        arduino.reset_input_buffer()
        arduino.write(b'2')

```

```

        arduino.flush()

        time.sleep(0.2) # brief pause for Arduino to process
    else:
        print("⚠ No MPU data collected, skipping motion check.")
        arduino.write(b'2')
        arduino.flush()

    else:
        print("❌ Unauthorized card.")
        arduino.write(b'2')
        arduino.flush()

    time.sleep(0.2) # allow Arduino to process each command

except KeyboardInterrupt:
    print("\nExiting...")
    arduino.close()
    break

```

4.4 Control Algorithm

1. Defining pins:
 - The Arduino is connected to the MPU6050 sensor via the I²C interface:
 - SDA → A4
 - SCL → A5
 - VCC → 5V
 - GND → GND

- The RFID module (MFRC522) is connected through the SPI interface:
 - SDA → D10
 - SCK → D13
 - MOSI → D11
 - MISO → D12
 - RST → D8
 - 3.3V → 3.3V
 - GND → GND
- A servo motor is attached to D10 (signal), 5V, and GND.
- Two LED indicators are connected:
 - Green LED → D11 (via 220Ω resistor)
 - Red LED → D9 (via 220Ω resistor)
- A USB cable connects the Arduino to the computer for serial communication and power supply.

2. Global variable:

- Several global variables are declared to store sensor data and control values:
 - int ax, ay, az → store accelerometer readings for the X, Y, Z axes.
 - float mean_ax, mean_ay → store baseline calibration values.
 - int motion_score → calculate average motion intensity.
- These variables are continuously updated to represent the real-time movement of the MPU6050.

3. Setup function:

- Serial.begin(9600) starts serial communication between the Arduino and the computer.
- Wire.begin() initializes I²C communication with the MPU6050 sensor.
- mpu.initialize() sets up the MPU6050 sensor.

- `mpu.testConnection()` checks the connection; if it fails, the message "MPU6050 connection failed!" is printed and the program stops.
- The servo motor is attached to pin 10 and moved to 0° (locked position).
- The green and red LEDs are set as output to indicate system status.

4. Main loop:

These actions are continuously carried out by the `loop()` function:

a. Read motion data:

- The function `mpu.getAcceleration(&ax, &ay, &az)` retrieves acceleration values from the MPU6050.
- These values are sent to the serial port as text in the format:
`AcX=<value>|AcY=<value>|AcZ=<value>`

b. Listen for serial commands:

- The Arduino waits for commands from the Python program via the serial interface.
- If command = '1' → Access Granted.
- If command = '2' → Access Denied.

c. Control actions:

- `ACCESS_GRANTED ('1')`
 - Turn green LED ON.
 - Move servo to 60° (unlock).
 - Wait 3 seconds.
 - Move servo back to 0° (lock).
 - Turn green LED OFF.
- `ACCESS_DENIED ('2')`

- Turn red LED ON for 2 seconds.
- Keep servo at 0° (locked).
- Turn red LED OFF.

d. Delay

- A short delay (50–100 ms) ensures consistent data transmission and prevents serial overflow.

5. Python Control Algorithm

Serial Connection:

- Establish a serial link on COM3 at 9600 baud using the PySerial library.

RFID Authentication:

- The program waits for the user to input an RFID card ID.
- If the scanned ID matches the authorized ID, the program proceeds to motion detection.
- Otherwise, it sends '2' to the Arduino (Access Denied).

Motion Detection:

- The system reads MPU6050 data for 2 seconds after authorization.
- Calculates the motion score by comparing live readings to baseline values.
- If motion score > threshold → motion detected → send '1' to Arduino.
- Otherwise → send '2'.

Access Feedback:

- Upon receiving '1', Arduino unlocks the servo and activates the green LED.
- Upon receiving '2', Arduino activates the red LED and remains locked.

Loop Operation:

- The process repeats continuously until manually stopped by the user.

5.0 DATA COLLECTION

UID	Accessible	Motion	Green LED	Red	Serve
0003855091	Granted	Yes	Turn On	Turn Off	Rotate 90° and turn back to 0°
0013084287	Denied	No	Turn Off	Turn On	Remains 0°

6.0 DATA ANALYSIS

- The MPU6050 sensor measured acceleration along the X, Y, and Z axes to detect motion.
- Arduino continuously sent raw acceleration data to the computer through serial communication.
- Python received this data and calculated the motion score using the formula:
$$\text{Motion Score} = \text{mean}(|A_{\text{current}} - A_{\text{baseline}}|)$$
- The motion score determined whether the user performed sufficient movement after scanning the RFID card.
- If the motion score exceeded the threshold, Python sent the command '1' (Access Granted) to Arduino.
- If the score was below the threshold or the card was unauthorized, it sent '2' (Access Denied).
- For Access Granted, the green LED turned ON, the servo rotated to 60° (unlock), and then returned to 0° after 3 seconds.
- For Access Denied, the red LED turned ON for 2 seconds and the servo remained locked at 0°.
- The authorized UID "0003855091" consistently unlocked the system, while other UIDs were rejected.

- Sensor readings were generally stable with minor noise, and serial communication between Arduino and Python was smooth.
- Overall, the system accurately detected motion and controlled access in real time with reliable responses across multiple trials.

7.0 RESULTS

Motion detection, servo control, and RFID authentication were all effectively integrated for the second experiment. After confirming RFID input and comparing it with the authorised ID that was recorded, the Python application examined motion activity using MPU6050 measurements. The solution instructed the Arduino to turn on the green LED and turn the servo motor to the unlocked position when an authorised RFID card was inserted and the sensor sensed a noticeable amount of movement. On the other hand, the red LED illuminated and the servo stayed locked when an unauthorised card was inserted or no motion was detected. With consistent motion detection based on acceleration variation from the baseline, the Arduino and Python real-time serial connection functioned dependably. The hardware (LEDs, servo, MPU6050) and software (Python-RFID control logic) in the system worked well together. Overall, the findings demonstrated a working prototype of a safe access control mechanism and validated that the integrated system can carry out two-step identification authentication and motion verification, before allowing access.

8.0 DISCUSSION

Expected vs Observed Outcomes

Aspect	Expected Outcome	Observed Outcome
MPU6050 sensor	Provides smooth and responsive motion data.	Worked as intended, with minor noise when unstable.
Python–Arduino link	Stable real-time data communication.	Communication was consistent with a slight delay.
RFID verification	Detects valid and invalid cards accurately.	Worked as intended.
Motion–RFID integration	Unlocks only with correct card and motion.	Worked as intended

Through this experiment, we managed to design and test a motion-activated RFID access system that combines hardware and software control. In Task 1, we focused on collecting and visualising motion data from the MPU6050 sensor using Arduino and Python. The real-time graph helped us understand how acceleration values change with movement, confirming that the sensor and serial communication worked properly.

In Task 2, we integrated the RFID module, servo motor, LEDs, and MPU6050 to form a complete smart access system. The RFID provided the first layer of authentication, while the motion gesture acted as the second layer. Using Python, we analysed the sensor data and sent commands to the Arduino to unlock or deny access based on both RFID verification and motion detection. The system worked well, showing clear responses when a valid card and correct motion were detected.

We did face some small issues, such as occasional delay in servo movement and inconsistent motion readings when the sensor was not stable. These could be improved with better filtering or calibration. Overall, this experiment helped us understand how sensors, actuators, and programming can work together in real-time automation systems.

9.0 CONCLUSION

In conclusion, this experiment was successful in developing a motion-activated RFID system that combined sensor data processing, real-time control, and multi-layer authentication. The integration between the MPU6050 sensor, RFID module, and servo motor through Arduino and Python worked effectively and demonstrated how different components can communicate to achieve an intelligent automation task.

Through this lab, we gained valuable hands-on experience in microcontroller interfacing, sensor calibration, data transmission, and actuator control. We also learned how Python can be used to process sensor data and control Arduino-based hardware, strengthening our understanding of serial communication in mechatronic systems.

Although there were minor challenges such as motion instability and occasional delay, the overall system achieved its objectives and functioned reliably. This project not only improved

our technical skills but also enhanced our teamwork, problem-solving, and system integration abilities, which are essential aspects in real-world mechatronic design and development.

10.0 RECOMMENDATIONS

- Improve sensor stability – Mount the MPU6050 securely to reduce vibration and noise.
- Add data filtering – Apply a simple low-pass or moving average filter to smooth out sensor readings.
- Enhance calibration – Include an auto-calibration step during startup to reduce drift.
- Stabilize power – Use a separate power source or capacitor for the servo to avoid voltage drops.
- Refine gesture detection – Use clearer thresholds or a more accurate method to detect motion patterns.
- Add user feedback – Use LEDs or a buzzer to indicate when motion detection or RFID scanning is in progress.
- Improve system reliability – Implement error handling for unstable readings or delayed servo responses.

11.0 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/>

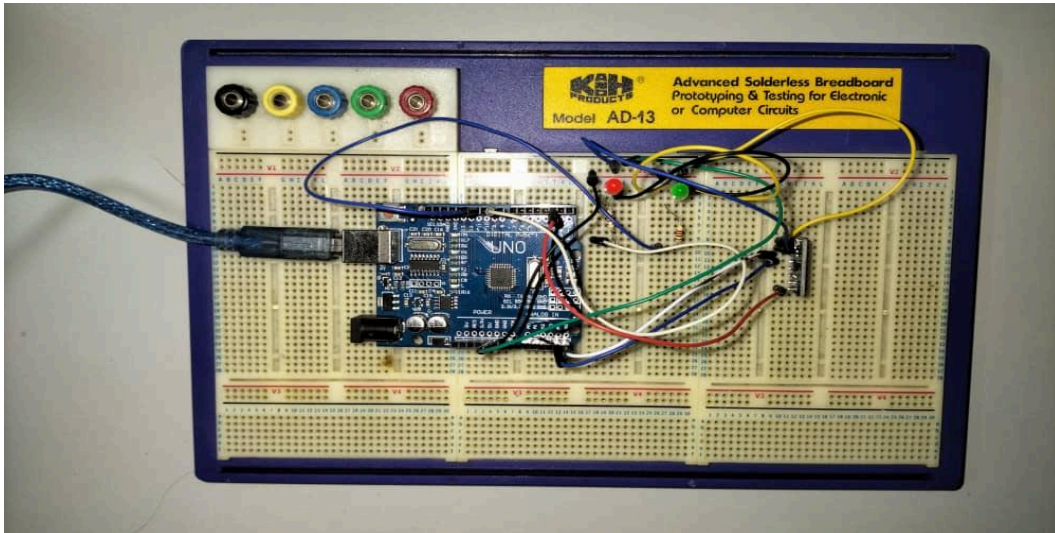
Random Nerd Tutorials. (n.d.). *Arduino time attendance system with RFID.*

Retrieved from <https://randomnerdtutorials.com/arduino-time-attendance-system-with-rfid/>

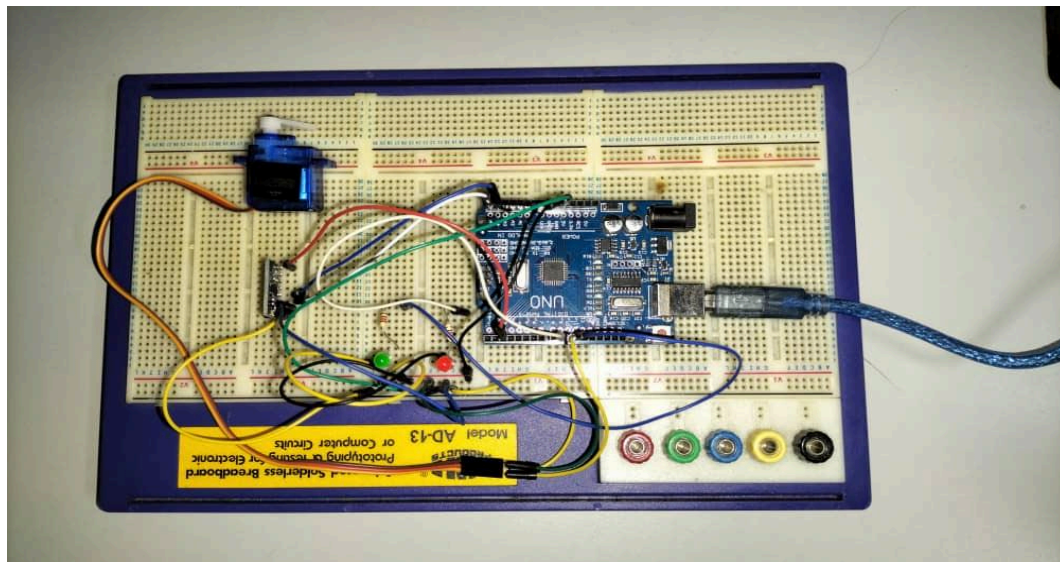
Instructables. (n.d.). *Arduino + MFRC522 RFID reader.*

Retrieved from <https://www.instructables.com/Arduino-MFRC522-RFID-READER/>

12.0 APPENDICES



Circuit Design Task 1



Circuit Design Task 2

13.0 ACKNOWLEDGEMENTS

Hereby, we acknowledge the guidance provided by Dr Zulkifli with his impeccable knowledge in this field. We also thank Br. Harith with his assistance in correcting our work thus leading to a successful experiment. Not to forget our fellow colleagues from other groups that contributed in providing ideas that helped in achieving the same output together from this experiment.

14.0 STUDENTS DECLARATION

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: HARIZ IRFAN BIN MOHD ROZHAN

Matric Number: 2318583

Read [/]

Understand [/]

Agree [/]

Signature: 


Name: ABDULLAH HASAN BIN SIDEK

Matric Number: 2318817

Read [/]

Understand [/]

Agree [/]

Signature: 


Name: TAN YONG JIA

Matric Number: 2319155

Read [/]

Understand [/]

Agree [/]

Signature: 

Name: NUR QISTINA BINTI MOHD FAIZAL

Matric Number: 2319512

Read [/]

Understand [/]

Agree [/]