MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 1 2025/2026

WEEK 4: SERIAL INTERFACING WITH MICROCONTROLLER (SENSORS & ACTUATORS)

SECTION 1

GROUP 9

LECTURER: ZULKIFLI BIN ZAINAL ABIDIN & WAHJU SEDIONO

NO.	GROUP MEMBERS	MATRIC NO.
1.	AHMAD FIRDAUS HUSAINI BIN HASAN AL-BANNA	2315377
2.	ADIB ZAFFRY BIN MOHD ABDUL RADZI	2315149
3.	UMAR FAROUQI BIN ABU HISHAM	2314995

Date of Submission: 5 November 2025

ABSTRACT

This experiment focuses on designing a motion-activated RFID access control system by integrating multiple sensors and communication interfaces using an Arduino microcontroller. The setup incorporates an MPU6050 motion sensor, an RFID reader, a servo motor, and LED indicators to form an intelligent authentication mechanism. The MPU6050 detects specific motion patterns, while the RFID module verifies authorized users through unique identification (UID) tags. Communication between the Arduino and Python enables real-time motion visualization, serial data analysis, and coordinated system operation.

In the first phase, the MPU6050 was utilized to detect circular motion based on x–y coordinate data. Predefined hand movements were performed, and the resulting motion data was collected and visualized using Python to identify and classify circular motion patterns.

In the second phase, the RFID system was implemented to differentiate between authorized and unauthorized UIDs. When both correct RFID verification and valid motion patterns are detected, the servo motor activates by rotating, and the green LED illuminates. Conversely, if an invalid UID is detected, the red LED lights up to indicate access denial.

Overall, this integration highlights the practical implementation of serial communication, sensor data fusion, and real-time control in mechatronic systems. The experiment effectively demonstrates how combining sensor fusion and serial interfacing can enhance system security and automation in smart access control applications.

TABLE OF CONTENTS

1.0 INTRODUCTION

The objective of this experiment is to design and implement a motion-activated RFID access control system that integrates sensors and serial communication using an Arduino microcontroller. The project aims to develop a smart security mechanism that grants access only when a registered RFID tag is detected and a predefined motion pattern is correctly performed. Through this experiment, students gain practical experience in combining sensors, actuators, and software communication between Arduino and Python.

The MPU6050 motion sensor plays a key role in detecting and analyzing hand movements through its built-in accelerometer and gyroscope, which capture data along the x, y, and z axes. Meanwhile, the RFID module identifies authorized users using unique tag IDs. When both the correct motion and a valid RFID tag are recognized, the Arduino triggers the servo motor to rotate, simulating an unlocking action, and activates the green LED. If the RFID tag is invalid or the motion is incorrect, the system remains locked and the red LED lights up.

This experiment emphasizes the application of sensor fusion, serial communication, and automation principles within mechatronic systems. The expected outcome is a system capable of verifying both motion and identity before granting access, effectively demonstrating how the integration of motion sensing and RFID technology can enhance security and reliability in access control applications.

TASK 1

2.0 MATERIALS AND EQUIPMENTS

1. Arduino Uno
2. Breadboard
3. MPU6050 sensor
4. Computer with Arduino IDE and Python installed
5. Jumper wires
6. USB cable

3.0 EXPERIMENTAL SETUP

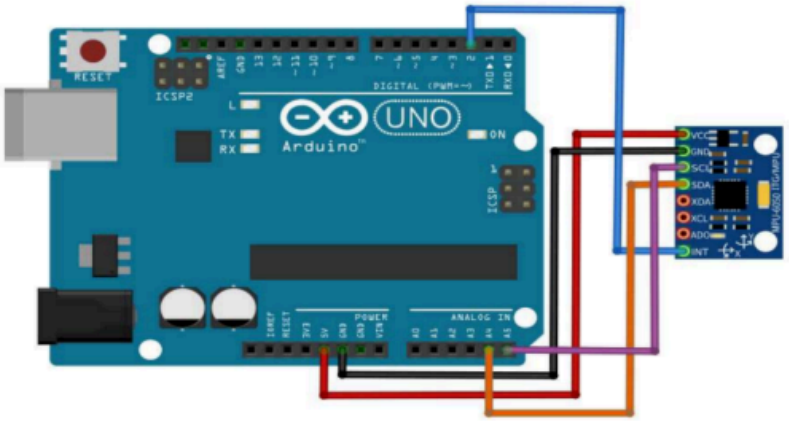


Fig. 1: Arduino-MPU6050 Connections

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 board to the computer via a USB cable.

3. Uploaded the Arduino program designed to collect data from the MPU6050 motion sensor.

4. Launched the Python application to receive and visualize the sensor data in real time.

5. Moved the MPU6050 sensor in various directions to evaluate its motion tracking performance.

6. Executed a circular hand motion and observed the corresponding pattern displayed on the graph.

7. Recorded and saved the obtained data for further analysis.

4.0 METHODOLOGY

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)

2. Programming Logic

a. 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

b. 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

c. 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

d. 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

3. Control Algorithm

1. **System Initialization**
    - Begin serial communication between the Arduino and the computer at a baud rate of 9600.
    - Initialize I2C communication using `Wire.begin()` to establish a connection with the MPU6050 sensor.
    - Configure and initialize the MPU6050 module.
2. **Sensor Data Acquisition (Arduino)**
    - Continuously read raw accelerometer (`ax`, `ay`, `az`) and gyroscope (`gx`, `gy`, `gz`) values from the MPU6050 using the `getMotion6()` function.
    - Format the accelerometer readings into a comma-separated string.
3. **Data Reception and Processing (Python)**
    - Continuously listen to the serial port for incoming data sent from the Arduino.
    - Read and decode each incoming line of data.
    - Split the comma-separated string into individual acceleration components (`ax_raw`, `ay_raw`, `az_raw`) for further analysis.
4. **Data Visualization Control (Python)**
    - Set up a real-time scatter plot using *matplotlib* to display acceleration data along the X and Y axes.
    - Continuously update the plot with the most recent acceleration readings.
    - Limit the visualization to the latest 100 data points to maintain smooth and responsive motion tracking.

4. Code Used

a. Arduino ide

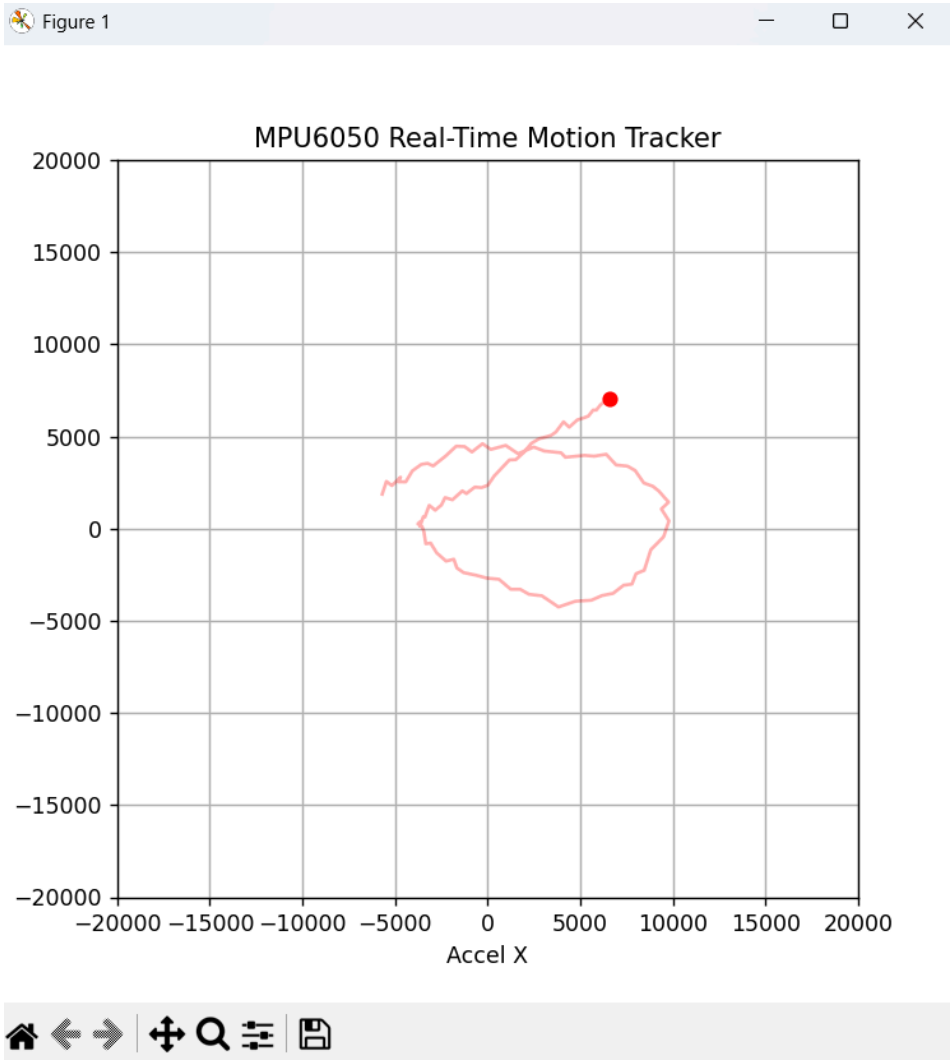
```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;

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

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

  // Send comma-separated accelerometer data
  Serial.print(ax); Serial.print(",");
  Serial.print(ay); Serial.print(",");
  Serial.println(az);

  delay(50);   // 20 samples per second
}
```

b. Pycharm

```
import serial
import matplotlib.pyplot as plt
import numpy as np
from collections import deque
import time
import warnings

# Suppress harmless divide-by-zero warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)

# === SETTINGS ===
PORT = 'COM4'          # Change this to your Arduino port
BAUD = 9600
MAX_POINTS = 100       # Number of points to keep on screen
SMOOTH_WINDOW = 5      # Moving average window
TRAIL_ALPHA = 0.3      # Trail transparency

# === SERIAL INIT ===
ser = serial.Serial(PORT, BAUD, timeout=1)
time.sleep(2)  # allow Arduino reset

# === PLOT INIT ===
plt.ion()
fig, ax = plt.subplots(figsize=(6, 6))
ax.set_xlim(-20000, 20000)
ax.set_ylim(-20000, 20000)
ax.set_xlabel('Accel X')
ax.set_ylabel('Accel Y')
ax.set_title('MPU6050 Real-Time Motion Tracker')
ax.grid(True)

# Main moving point (red)
point, = ax.plot([], [], 'ro', markersize=6)
# Fading trail (light red line)
trail, = ax.plot([], [], 'r-', alpha=TRAIL_ALPHA)

x_vals, y_vals = deque(maxlen=MAX_POINTS), deque(maxlen=MAX_POINTS)

# === Smoothing Function ===
def smooth(values, new_val, window=SMOOTH_WINDOW):
    """Return a moving average of recent values."""
    if not values:
        return new_val
    recent = list(values)[-window:] + [new_val]
    return np.mean(recent)

# === Circle Detection (Optional) ===
def detect_circle(x, y, threshold=0.7):
    if len(x) < 20:
        return False
    if np.std(x) < 1e-6 or np.std(y) < 1e-6:
        return False
    x_norm = (x - np.mean(x)) / (np.std(x) + 1e-9)
    y_norm = (y - np.mean(y)) / (np.std(y) + 1e-9)
    corr = np.corrcoef(x_norm, np.roll(y_norm, 5))[0, 1]
    return corr > threshold
```

```
print(f"Reading from {PORT}...")
print("Press Ctrl + C to stop.")

# === Main Loop ===
try:
    while True:
        raw =
ser.readline().decode(errors='ignore')
.strip()
        if ',' in raw:
            vals = raw.split(',')
            if len(vals) >= 2:
                try:
                    ax_val = int(vals[0])
                    ay_val = int(vals[1])

                    # Smooth values
                    ax_smooth =
smooth(x_vals, ax_val)
                    ay_smooth =
smooth(y_vals, ay_val)

                    # Append to deque

x_vals.append(ax_smooth)

y_vals.append(ay_smooth)

                    # Update trail and
moving point
                    trail.set_data(x_vals,
y_vals)

point.set_data([x_vals[-1]],
[y_vals[-1]])

                    plt.draw()
                    plt.pause(0.01)

                    # Detect circular motion
                    if
detect_circle(np.array(x_vals),
np.array(y_vals)):
                        print("🔵 Circular
motion detected!")

                except ValueError:
                    pass

except KeyboardInterrupt:
    print("\nStopped by user.")

finally:
    ser.close()
    plt.ioff()
    plt.show()
    print("Serial connection closed.")
```

5.0 DATA COLLECTION



6.0 DATA ANALYSIS

In this experiment, the MPU6050 motion sensor was utilized to monitor real-time movement and visualize it on a computer interface. The sensor continuously provided acceleration readings along the x, y, and z axes, which varied according to the sensor's orientation and motion. When held stationary, the readings remained relatively constant, indicating stable sensor performance. However, once the sensor was moved in a circular trajectory; noticeable fluctuations appeared in the x and y axes.

From the plotted data, the trajectory exhibits a closed-loop pattern, approximating a circular or slightly elliptical shape, which validates that the circular motion gesture was successfully captured.

Overall, the MPU6050 sensor proved to be highly responsive and reliable in detecting hand movements. The plotted trajectory corresponded closely with the actual motion performed, validating the consistency of the sensor's readings.

7.0 RESULTS

The experiment successfully demonstrated real-time motion tracking using the MPU6050 accelerometer and gyroscope sensor. Data collected from the sensor's X, Y, and Z axes were transmitted to the computer via the Arduino serial interface and visualized dynamically using Python and Matplotlib.

- The MPU6050 effectively measured real-time acceleration in multiple axes.
- The Python visualization successfully represented the sensor's motion path dynamically.
- The circular gesture produced a distinct looping pattern on the plot, validating correct data capture.

- The readings were consistent and repeatable, indicating reliable communication and stable sensor output.

8.0 DISCUSSION

The experiment successfully demonstrated the process of capturing real-time motion data using the MPU6050 accelerometer and gyroscope sensor. By connecting the sensor to the Arduino Uno through the I²C interface, acceleration values along the X, Y, and Z axes were transmitted continuously to the computer through serial communication. The Python program was then used to visualize the sensor output in a live two-dimensional plot, providing a clear representation of the motion path.

During the experiment, movement of the sensor produced visible changes in the plotted data, confirming that the sensor accurately detected and transmitted acceleration values. When a circular motion was performed, the plot displayed a recurring pattern, proving that circular motion can be detected and analyzed through simple correlation between X and Y acceleration data. Minor fluctuations were observed due to noise and hand instability, which could be minimized using smoothing or filtering techniques in future improvements.

Overall, Task 1 successfully met its objectives by demonstrating serial communication between the Arduino and computer, real-time data plotting in Python, and basic motion pattern recognition.

9.0 CONCLUSION

In conclusion, the MPU6050 module was successfully interfaced with the Arduino Uno to measure and visualize motion data in real time. The serial communication between Arduino and Python enabled dynamic tracking of acceleration values and effective motion pattern

visualization. The system was able to detect and display circular motion accurately, validating the reliability of the sensor and the efficiency of the communication setup. This task provided valuable understanding of sensor calibration, data visualization, and real-time motion tracking.

10.0 RECOMMENDATIONS

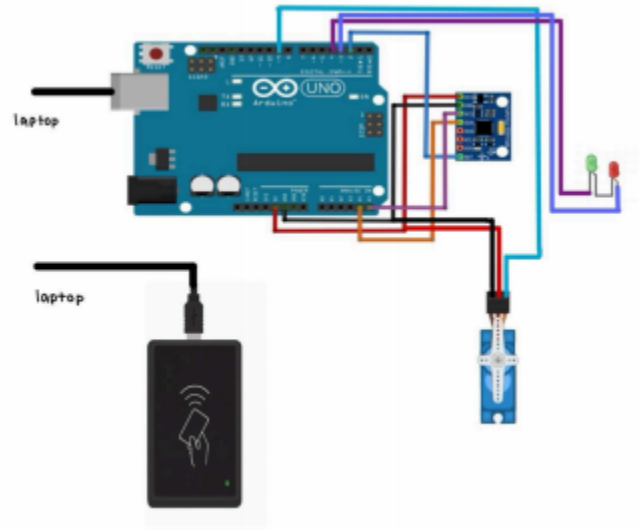
It is recommended to include data smoothing and filtering algorithms such as a moving average or low-pass filter to reduce sensor noise. Calibration procedures should be conducted before data collection to improve accuracy and consistency. Future versions of the experiment could incorporate gyroscope data to analyze orientation or rotational movement in three dimensions. Additionally, integrating a graphical interface would enhance data interpretation and overall system usability.

TASK 2 : INTEGRATED SMART ACCESS SYSTEM

11.0 MATERIALS AND EQUIPMENTS

1. Arduino Uno
2. RFID card reader with USB
3. MPU6050 Accelerometer and Gyroscope Module
4. SG90 Servo Motor
5. Green LED and Red LED
6. Computer with Arduino IDE and Python installed
7. 220 Ω Resistor
8. Jumper Wires
9. Breadboard
10. RFID Card 11. USB Cable

12.0 EXPERIMENTAL SETUP



1. Connected the RFID reader directly to the laptop.

2. Linked the servo motor’s signal pin to digital pin D9 on the Arduino.

3. Connected the green LED to D4 and the red LED to D3, each with an appropriate

current-limiting resistor.

4. Reused the MPU6050 sensor setup from Experiment 4A.

5. Connected the Arduino board to the computer via a USB cable.

6. Uploaded the Arduino program responsible for controlling the servo motor and LEDs based on serial commands (‘A’ for access granted and ‘D’ for access denied).

7. Executed the Python program to read RFID tags and verify authorized IDs.

8. Performed the required circular hand motion after tapping the RFID card to confirm motion recognition.

9. Observed the system behavior:
   - Green LED illuminated and servo unlocked for a valid card combined with the correct motion.
   - Red LED illuminated for an invalid card or incorrect motion.

10. Documented and saved the experimental results for further analysis.

13.0 METHODOLOGY

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)

2. Programming Logic

a. 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.

b. RFID Verification

- The Python program waits for RFID input from a keyboard-based RFID reader.
- It compares the scanned RFID tag data against a predefined list of authorized card IDs.
- If the card is recognized as authorized, the program sends the signal ‘A’ to the Arduino.
- If the card is not authorized, it sends the signal ‘D’ to the Arduino.
- The console displays feedback messages such as “Card verified” for access granted

c. 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.

d. Circular Motion Detection (IMU Sensor)

● Once the RFID is verified, the Arduino begins detecting circular motion using data from the MPU6050 accelerometer.
● It continuously reads acceleration values along the X and Z axes.
● The motion pattern is analyzed based on four quadrant checkpoints — right, left, up, and down.
● If all four checkpoints are completed within a 5-second timeout, the system confirms successful circular motion detection.

e. 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.

3. Code Used

a. Arduino IDE

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

MPU6050 mpu;
Servo myServo;

const int GREEN_LED = 4;
const int RED_LED = 3;
const int SERVO_PIN = 9;

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

  pinMode(GREEN_LED,
OUTPUT);
  pinMode(RED_LED, OUTPUT);

  myServo.attach(SERVO_PIN);
  myServo.write(0);

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

  Serial.println("System Ready");
}

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

    switch (cmd) {

      case 'A':  // Motion detected and
authorized
        digitalWrite(RED_LED, LOW);
        digitalWrite(GREEN_LED,
HIGH);
        myServo.write(60);
        delay(5000);
        digitalWrite(GREEN_LED,
LOW);
        myServo.write(0);
        break;

      case 'D':  // No motion or
unauthorized
        digitalWrite(GREEN_LED,
LOW);
        digitalWrite(RED_LED,
HIGH);
        myServo.write(0);
        delay(3000);
        digitalWrite(RED_LED, LOW);
        break;

      case 'M':  // MPU6050 motion
data request
        int16_t ax, ay, az, gx, gy, gz;
        mpu.getMotion6(&ax, &ay,
&az, &gx, &gy, &gz);
        Serial.print(ax);
Serial.print(",");
        Serial.print(ay);
Serial.print(",");
        Serial.print(az);
Serial.print(",");
        Serial.print(gx);
Serial.print(",");
        Serial.print(gy);
Serial.print(",");
```

```
      Serial.println(gz);
      break;
    }
```

```
  }
}
```

b. Pycharm

```
import serial
import time
import numpy as np

# === Configuration ===
arduino = serial.Serial("COM4",
9600, timeout=1)
time.sleep(2)

AUTHORIZED_ID = "0013069760"

print("RFID + Motion Access
System Running...\n")

def read_mpu_data(samples=50):
    """Collects motion data from
Arduino"""
    ax_list, ay_list, az_list, gx_list,
gy_list, gz_list = [], [], [], [], [], []

    for _ in range(samples):
        arduino.write(b'M')  # Request
MPU6050 data
        line =
arduino.readline().decode().strip()
        if line:
            try:
                ax, ay, az, gx, gy, gz =
map(int, line.split(','))
                ax_list.append(ax)
                ay_list.append(ay)
                az_list.append(az)
                gx_list.append(gx)
                gy_list.append(gy)
                gz_list.append(gz)
            except:
                continue
        time.sleep(0.05)

    return np.array(gx_list),
np.array(gy_list), np.array(gz_list)

def detect_circular_motion(gx, gy,
gz):
    """Simple heuristic: large
variations in gyro data indicate
motion"""
    threshold = 3000  # adjust based
on real motion
    variation = np.std(gx) + np.std(gy)
+ np.std(gz)
    print(f"Motion Variation Score:
{variation:.2f}")
    return variation > threshold

# === Main Loop ===
while True:
    try:
        card = input("Tap RFID card:
").strip()
        print("Card Scanned:", card)

        if card == AUTHORIZED_ID:
```

```
            print("✅ Authorized. Please perform a circular motion...")
            gx, gy, gz = read_mpu_data(samples=60)

            if detect_circular_motion(gx, gy, gz):
                print("🌀 Motion detected → Access Granted.")
                arduino.write(b'A')
            else:
                print("⚠️ No circular motion detected → Access Denied.")
                arduino.write(b'D')

        else:
            print("❌ Unauthorized Card.")
            arduino.write(b'D')

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

14.0 DATA COLLECTION

AUTHORIZED_ID = "0013069760"

Authorized UID	Circular Motion	Accessible	Green LED	Red LED	Servo
Yes	Yes	Granted	Turn On	Turn Off	Rotates 90°
Yes	No	Denied	Turn Off	Turn On	Remains 0°
No	-	Denied	Turn Off	Turn On	Remains 0°

15.0 DATA ANALYSIS

When an RFID tag was scanned, the MFRC522 module read the UID and transmitted it to the Python control script through the serial port. The Python program compared the UID with a list of authorized IDs stored in the program memory.

- Authorized tags triggered a motion verification request.
- Unauthorized tags immediately triggered access denial.

Once a valid RFID tag was scanned, the system prompted the user to perform a circular hand gesture while holding the MPU6050 sensor. The accelerometer and gyroscope data were analyzed to determine whether the motion matched a circular trajectory pattern.

If the motion was executed correctly, the Python program sent a command to the Arduino to unlock the servo motor and activate the green LED. Conversely, if an unregistered tag was used or the motion was incorrect, the servo motor remained locked and the red LED was turned on.

16.0 RESULTS

1. RFID Identification:

   The RFID module effectively detected and retrieved the unique identifiers (UIDs) of the tested RFID tags. Each UID was successfully transmitted to the connected Python program through serial communication for authentication purposes.

2. Access Verification:

   The Python program accurately matched the received UIDs against a predefined list of authorized entries. Authorized tags were allowed to proceed to the motion detection stage, while unauthorized tags immediately triggered an access denial response. This process demonstrated reliable integration between the RFID reader and the software-based verification system.

3. System Operation:

   Upon successful validation and correct circular motion detection, the system executed the unlock procedure; rotating the servo motor and activating the green LED to indicate access granted. In cases of invalid tags or incorrect gestures, the servo remained in its locked position, accompanied by the red LED indicator. Overall, the system displayed consistent performance, delivering rapid and accurate feedback for each test scenario.

17.0 DISCUSSION

The integration of the MPU6050 sensor with the servo motor and LED system successfully demonstrated a basic smart access control mechanism based on motion detection. The system used the MPU6050 to identify hand motion, which then triggered a servo motor to simulate door unlocking when motion was detected. The use of LEDs provided clear visual feedback, where the green LED indicated access granted and the red LED indicated access denied.

Throughout the experiment, the Arduino successfully processed acceleration data from the MPU6050 and controlled the servo output accordingly. The serial communication between the Arduino and Python provided synchronization between motion detection and system response. However, small delays and false triggers were observed due to sensor sensitivity and serial communication latency. These limitations can be minimized through improved threshold calibration and optimized serial handling in the code.

Overall, the system effectively demonstrated how motion sensing can be integrated into automated control systems for secure and responsive access management.

18.0 RECOMMENDATIONS

For improved performance, it is recommended to use additional filtering to minimize noise and false motion detection. The servo motor should be powered using a dedicated 5V external supply to ensure stable movement. The system could be enhanced further by including a password, keypad, or RFID module for multi-factor access verification. Using a higher-performance microcontroller or integrating wireless communication would also allow remote access monitoring and better system flexibility.

19.0 CONCLUSION

In conclusion, the developed system successfully integrated the MPU6050 sensor with the Arduino Uno to detect motion and activate a servo lock mechanism. The experiment demonstrated an effective link between sensor input and actuator output, supported by real-time serial communication. The use of motion as an access condition shows the potential of motion-based authentication in smart control systems. Despite minor challenges with sensitivity

and timing, the system met its objectives and proved functional as a prototype for motion-controlled access.

20.0 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/

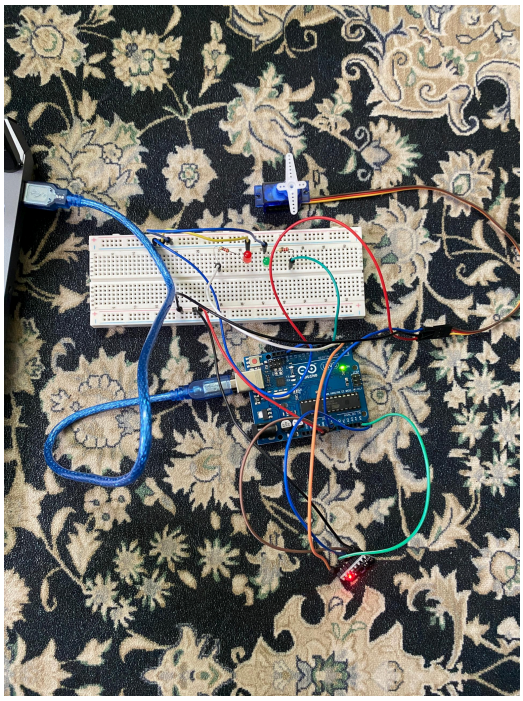
Santos, R. (2016, March 23). *MFRC522 RFID Reader with Arduino Tutorial | Random Nerd Tutorials*. Random Nerd Tutorials. https://randomnerdtutorials.com/security-access-using-mfrc522-rfid-reader-with-arduino/

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

21. APPENDICES

TASK 1

TASK 2



22.0 ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Dr. Zulkifli Bin Zainal Abidin and Dr. Wahju Sediono for their continuous support and for providing the essential resources and facilities required to carry out this project. We also extend our heartfelt appreciation to everyone who contributed to the success of this lab report through their valuable insights, assistance, and feedback.

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 acknowledgement, and that the original work contained herein have 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 that no further improvement on the reports is needed from any of the individual contributors to the report.

Signature:

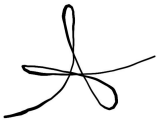
Read [/]

Name: AHMAD FIRDAUS HUSAINI BIN HASAN AL-BANNA
Matric Number: 2315377

Understand [/]
Agree [/]

Signature:

Read [/]



Name: ADIB ZAFFRY BIN MOHD ABDUL RADZI
Matric Number: 2315149

Understand [/]
Agree [/]

Signature:

Read [/]

Name: UMAR FAROUQI BIN ABU HISHAM
Matric Number: 2314995

Understand [/]
Agree [/]