MECHATRONICS SYSTEM INTEGRATION

MCTA3202

EXPERIMENT 4: TASK 1 & TASK 2

Serial interfacing with microcontroller:

Sensors and actuators

DATE: 27TH OCTOBER 2025
SECTION: 2
GROUP: 18
SEMESTER 1, 2025/2026

NO	NAME	MATRIC NO
1.	SHAMSUL HAKIMEE BIN SHAMSUL HAIREE	2315027
2.	QASHRINA AISYA BINTI MOHD NAZARUDIN	2315184
3.	NUREL HUMAIRAH BINTI MUHAMMAD FIRDAUS	2315680

DATE OF SUBMISSION:
Monday, 3TH November 2025

Abstract

This experiment integrates two subsystems, which are a motion-based gesture recognition system using the MPU6050 sensor and an RFID access control mechanism equipped with LEDs and a servo motor. The first subsystem detects and visualizes predefined hand gestures through real-time accelerometer and gyroscope data plotted using Python. The second subsystem authenticates RFID tags, verifying user credentials and providing corresponding LED and servo feedback. A green LED and servo actuation denote successful access, while a red LED indicates denial. Results demonstrated accurate gesture classification, reliable tag validation, and smooth actuator response. The project effectively showcases real-time sensor communication, data integration, and user feedback control, serving as a foundation for future advancements in smart access systems using machine learning and improved authentication techniques.

TABLE OF CONTENTS

1.0 Introduction

This experiment focuses on developing and integrating two embedded systems:

1. A hand gesture recognition module employing the MPU6050 accelerometer-gyroscope sensor
2. An RFID-based access control unit enhanced with LED indicators and servo actuation.

The objective is to demonstrate how microcontrollers can interface with sensors and external systems for real-time motion tracking and secure access management. The gesture system collects and processes motion data to classify basic hand movements, while the RFID system reads and validates unique identifiers (UIDs). Together, these systems illustrate the core principles of sensor fusion, data processing, and feedback-driven control, which are vital in applications such as human-computer interaction, automation, and intelligent security systems.

2.0 Materials and Equipment

2.1 Hand Gesture System Components

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

2.2 RFID System Components

1. Arduino board
2. RFID card reader with USB connectivity
3. RFID tags or cards for authentication
4. Servo Motor to control the angle
5. Jumper wires and Breadboard
6. Red and green LEDs
7. USB cables to connect the Arduino board and the RFID reader to the computer.
8. RS232 to USB adapter cable
9. Computer with Arduino IDE and Python installed

3.0 Experimental Setup

3.1 Task 1: Hand Gesture System of Real-Time Motion on Tracking Using MPU6050 Sensor

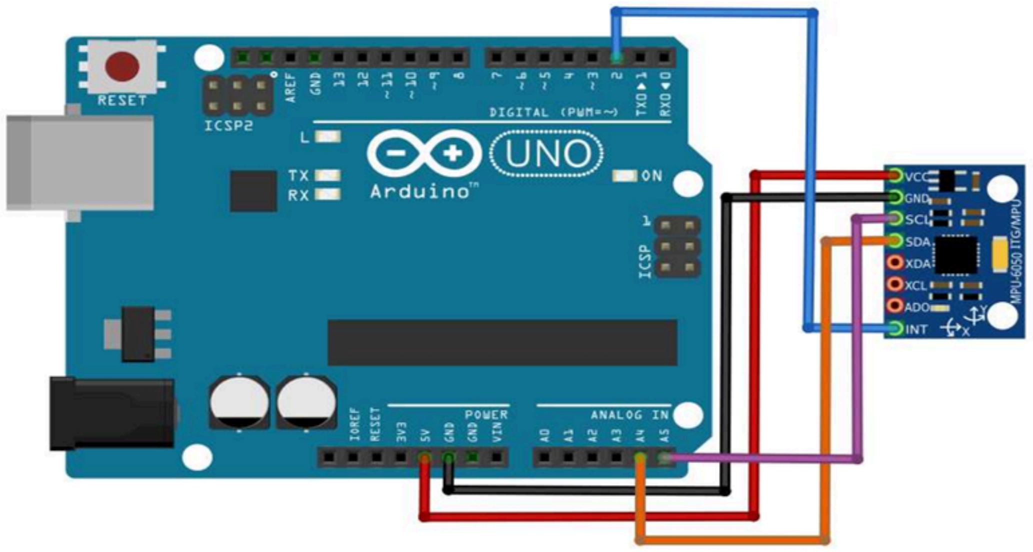


Figure 1.0

1. The MPU6050 sensor was connected to the Arduino via the I2C interface using the SCL and SDA pins.
2. The power (VCC) and ground (GND) of the MPU6050 were connected to the Arduino’s 5V and GND pins, respectively.
3. The Arduino board was connected to the PC via a USB cable.
4. Sensor data was transmitted to the PC via the serial port for visualization and gesture classification using Python.
5. The collected accelerometer and gyroscope data were processed and visualized using Matplotlib, where the detected gestures such as circular hand movements were plotted in a graphical form to observe the trajectory and accuracy of the motion.

3.2 Task 2: RFID + Servo Access System and Integrated Smart Access System

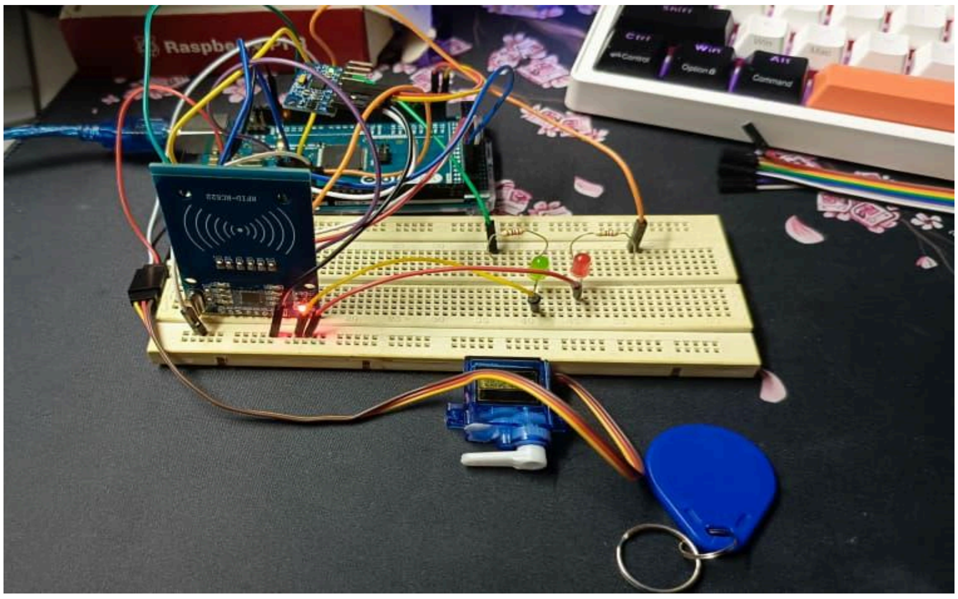
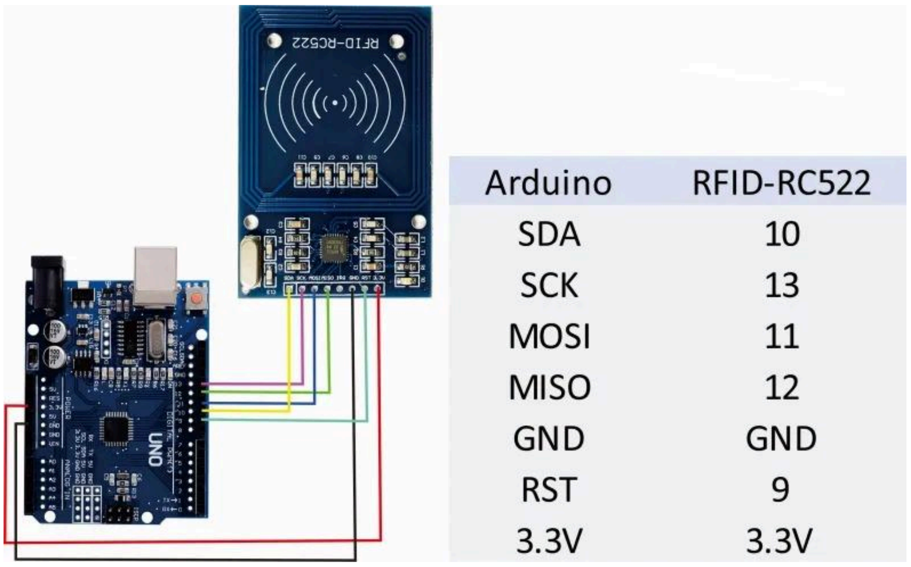


Figure 1.1



Arduino	RFID-RC522
SDA	10
SCK	13
MOSI	11
MISO	12
GND	GND
RST	9
3.3V	3.3V

Figure 1.2

1. The RFID-RC522 module was connected to the Arduino UNO using the SPI interface as shown in Figure 1.2.
2. The COM port of the RFID reader was verified using the Arduino Serial Monitor or Device Manager.
3. The entire circuit was assembled according to the schematic diagram shown in Figure 1.0.
4. The Arduino board was connected to the PC via USB for programming and serial communication.
5. A UID database was created and stored in a structured JSON file format

6. The Arduino code was uploaded, and Python was used to run the serial interface for UID detection, LED activation, and servo angle control.
7. Python was used to run the **serial interface**, which detected the card UID transmitted by the Arduino and triggered specific actions such as **LED activation** and **servo motor control** based on the recognized card.

4.0 METHODOLOGY

4.1 Hand Gesture System

a) Circuit Assembly

i) The MPU6050 sensor was connected to the Arduino via the I2C interface using the SCL and SDA pins on the Arduino Mega board.

ii) Meanwhile the INT pin was connected to (D2) Pin 2 on Arduino Mega.

iii) The power (VCC) and ground (GND) of the MPU6050 were connected to the Arduino's 5V and GND pins, respectively.

iv) The Arduino board was connected to the PC via a USB cable.

b) Programming Logic

__**Python**__

i) Import necessary libraries:
   1) **serial** for reading data from Arduino.
   2) **matplotlib.pyplo**t for plotting and visualising the motion trajectory in real time.
   3) **collections.deque** for storing a limited number of recent data points.
   4) **time** for handling delays and timing operations such as calibration duration.
   5) **numpy** for numerical operations such as calculating calibration offsets.
   6) **matplotlib.widgets.Button** for adding a reset button to the plot interface.

ii) Establish a serial connection with the Arduino on COM6 at 9600 baud rate.
   1) Briefly wait to stabilize the connection and discard a few initial readings.

iii) Calibrate the sensor:
   1) Keep the MPU6050 still for 3 seconds while collecting multiple acceleration samples.
   2) Compute and store the average offset for the X and Y axes to remove noise and bias.

iv) Initialize real-time plotting using matplotlib:
   1) Create a live plot window with labeled X and Y axes.
   2) Set square aspect ratio, grid lines, and defined plot limits.
   3) Add a **Reset Position** button to clear position and velocity data during execution.

v) Continuously read serial data from Arduino:
   1) Decode each incoming line, split values by commas, and extract acceleration data (ax, ay, az).
   2) Subtract calibration offsets to get corrected accelerations.
   3) Integrate acceleration to compute velocity and position.
   4) Apply damping to reduce drift and maintain plot stability.

vi) Handle errors and program termination safely:
   1) Catch serial or data decoding errors without crashing.
   2) On user interruption, close the serial port and keep the final plot displayed.

<u>Arduino</u>

i) Import necessary libraries:
   1) **Wire.h** for I2C communication between Arduino and MPU6050.
   2) **Adafruit_MPU6050.h** to handle sensor initialization and data retrieval.
   3) **Adafruit_Sensor.h** for standardized sensor data structure and readings.

ii) Create an MPU6050 object:
   1) **Adafruit_MPU6050** can create an instance to communicate with the sensor and access its data.

iii) Initialize Serial communication:
   1) **Serial.begin(9600)** will start the serial interface at 9600 baud rate to send data to a computer.

iv) Initialize I2C communication:
   1) **Wire.begin()** will start the I2C bus, allowing communication with the MPU6050.

v) Check MPU6050 connection:
   1) **mpu.begin()** will test if the MPU6050 is connected properly.
   2) If not detected, the program prints an error message and stops execution using **while(1);**.

vi) Display readiness message:
   1) Prints **“MPU6050 Ready. Sending accelerometer data...”** to indicate the sensor setup is complete.

vii) Read sensor data repeatedly in the loop:
   1) **mpu.getEvent(&a, &g, &temp)** will read current acceleration, gyroscope, and temperature data.

2) Only accelerometer values (x, y, z) are used and printed via serial.

viii) Add a short delay:

1) **delay(100)** was used to pause for 100ms before taking the next reading, resulting in 10 readings per second

c) Code Used

Python

```python
import serial

import time

import matplotlib.pyplot as plt

from collections import deque

import numpy as np

from matplotlib.widgets import Button

# --- Import type hints to satisfy the linter ---

from matplotlib.figure import Figure

from matplotlib.axes import Axes

from matplotlib.lines import Line2D

from typing import List, Deque


# --- Configuration ---

# !! CHANGE THIS to your Arduino's COM port

SERIAL_PORT = 'COM5'

BAUD_RATE = 9600

DT = 0.1  # Time step in seconds (matches Arduino delay(100))


# We will store the last N points to keep the plot clean

MAX_POINTS = 100

x_pos: Deque[float] = deque(maxlen=MAX_POINTS)  # Use deque for efficient
pop/append
```

```
y_pos: Deque[float] = deque(maxlen=MAX_POINTS)


# --- Physics Variables ---

vel_x, vel_y = 0.0, 0.0

pos_x, pos_y = 0.0, 0.0

ax_offset, ay_offset = 0.0, 0.0

# --- INCREASED DAMPING TO FIGHT DRIFT ---

DAMPING = 0.90  # Factor to reduce velocity drift (closer to 1 = less
damping)


# --- Type hint for ser ---

ser: serial.Serial


print("Connecting to Arduino...")

try:

   ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)

   time.sleep(2)  # Wait for serial connection


   # Read and discard the first few lines

   for _ in range(5):

      ser.readline()


   print(f"Successfully connected to {SERIAL_PORT}.")


except serial.SerialException as e:

   print(f"Error: Could not open serial port {SERIAL_PORT}.")

   print(f"Details: {e}")

   print("Please check the port and ensure Arduino IDE's Serial Monitor is
closed.")
```

```
   exit()


# --- CALIBRATION ---

print("\n--- CALIBRATION ---")

print("Keep the sensor perfectly still for 3 seconds...")

calib_x: List[float] = []

calib_y: List[float] = []

start_time = time.time()

while time.time() - start_time < 3.0:

   try:

      line_data = ser.readline().decode('utf-8').strip()

      if line_data:

            parts = line_data.split(',')

            if len(parts) == 3:

               calib_x.append(float(parts[0]))

               calib_y.append(float(parts[1]))

   # --- Fix for "Too broad exception" warning ---

   except (UnicodeDecodeError, ValueError, IndexError,
serial.SerialException) as e:

      print(f"Warning during calibration: {e}")

      continue  # Ignore errors during calibration


if not calib_x or not calib_y:

   print("Calibration failed! No data received. Exiting.")

   ser.close()

   exit()


ax_offset = np.mean(calib_x)
```

```python
ay_offset = np.mean(calib_y)

print(f"Calibration complete. Offset X: {ax_offset:.2f}, Offset Y:
{ay_offset:.2f}")

print("You can now move the sensor.\n")

# --- END CALIBRATION ---




# --- Set up the plot ---

plt.ion()  # Turn on interactive mode

# --- Add type hints for fig and ax to fix linter warnings ---

fig: Figure

ax: Axes

fig, ax = plt.subplots()

line: Line2D

line, = ax.plot(list(x_pos), list(y_pos), 'r-')


# Set plot limits - position is relative, so we start at a small range

# --- Specify named parameters to clear linter confusion ---

# --- INCREASED THE SCALE ---

ax.set_xlim(left=-1.0, right=1.0)

ax.set_ylim(bottom=-1.0, top=1.0)

ax.set_aspect('equal', 'box')  # Make the plot square


ax.set_xlabel("X-Position (m)")

ax.set_ylabel("Y-Position (m)")

ax.set_title("Real-Time MPU6050 Position Plot")

ax.grid()
```

```
# Add a "Reset Position" button

# --- Fix for "unused 'event'" warning ---

def reset_position(_event):

   """Callback function to reset the plot and physics variables."""

   global pos_x, pos_y, vel_x, vel_y

   pos_x, pos_y, vel_x, vel_y = 0.0, 0.0, 0.0, 0.0

   x_pos.clear()

   y_pos.clear()

   print("--- Position Reset ---")




# --- Fix for linter warning by using a TUPLE () instead of a LIST [] ---

ax_button: Axes = plt.axes((0.8, 0.0, 0.15, 0.05))  # [left, bottom, width,
height]

reset_btn = Button(ax_button, 'Reset')

reset_btn.on_clicked(reset_position)



try:

   while True:

      try:

            line_data = ser.readline().decode('utf-8').strip()

      except (UnicodeDecodeError, serial.SerialException):

            # This can happen if the Arduino disconnects

            print("Warning: Skipping bad serial data.")

            continue


      if line_data:
```

```python
try:

# We are receiving 3 values (ax, ay, az)

parts = line_data.split(',')

if len(parts) == 3:

    # 1. Get calibrated acceleration

    ax_calibrated = float(parts[0]) - ax_offset

     ay_calibrated = float(parts[1]) - ay_offset


    # 2. Integrate to get velocity

    vel_x += ax_calibrated * DT

    vel_y += ay_calibrated * DT


    # 3. Apply damping to velocity to fight drift

    vel_x *= DAMPING

    vel_y *= DAMPING


    # 4. Integrate to get position

    pos_x += vel_x * DT

    pos_y += vel_y * DT


    # Append new position data

    x_pos.append(pos_x)

     y_pos.append(pos_y)


    # Update the plot data

    line.set_xdata(list(x_pos))

     line.set_ydata(list(y_pos))
```

```
                # --- Auto-scaling the plot ---

                # --- Add comments to suppress "typo" warnings ---

                # noinspection PyTypeChecker

                xlims = ax.get_xlim()  # linter:ignore

                # noinspection PyTypeChecker

                ylims = ax.get_ylim()  # linter:ignore


                if pos_x > xlims[1] or pos_x < xlims[0] or \

                        pos_y > ylims[1] or pos_y < ylims[0]:

                    ax.relim()

                    ax.autoscale_view()


                # Redraw the canvas

                fig.canvas.draw()

                 fig.canvas.flush_events()


            else:

                print(f"Skipping malformed data: {line_data}")


            except (ValueError, IndexError):

            print(f"Skipping invalid data: {line_data}")


        plt.pause(0.01)  # Short pause to allow plot to update


except KeyboardInterrupt:

    print("\nProgram terminated by user.")

except Exception as e:

    # Catch other potential crashes (like plot window being closed)
    print(f"\nAn unexpected error occurred: {e}")

finally:

    if 'ser' in locals() and ser.is_open:

        ser.close()

        print("Serial port closed.")

    plt.ioff()  # Turn off interactive mode

    plt.show()  # Keep the final plot window open

    print("Plot window closed.")
```

Arduino

```
/*

  This is the simple Arduino sketch for Task 1.

  It ONLY reads the MPU6050 and prints the 3 accelerometer

  axes (ax, ay, az) to the Serial port, separated by commas.

*/



#include <Wire.h>

// We use the Adafruit library, which is better than the PDF's
MPU6050.h

#include <Adafruit_MPU6050.h>

#include <Adafruit_Sensor.h>



Adafruit_MPU6050 mpu;



void setup() {

  Serial.begin(9600);


  // Init I2C (for MPU6050)

  Wire.begin();

  if (!mpu.begin()) {

      Serial.println("MPU6050 connection failed! Check wiring.");

      while (1); // Stop here

  }



  // Set MPU ranges

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);

  mpu.setGyroRange(MPU6050_RANGE_500_DEG);

  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);



  Serial.println("MPU6050 Ready. Sending accelerometer data...");

}



void loop() {

  sensors_event_t a, g, temp;

  mpu.getEvent(&a, &g, &temp);



  // Send only the 3 accelerometer axes, as per the PDF's example

  Serial.print(a.acceleration.x);

  Serial.print(",");

  Serial.print(a.acceleration.y);

  Serial.print(",");

  Serial.print(a.acceleration.z);
  Serial.println();



  delay(100); // Send data 10 times per second

}
```

4.2 RFID System

a) Circuit Assembly

  i) RFID Reader (MFRC522):
    1) SDA pin connected to **digital pin 10** on the Arduino Mega.
    2) RST pin connected to **digital pin 9**.
    3) SCK pin connected to **pin 52 (SPI Clock)**.
    4) MOSI pin connected to **pin 51 (SPI MOSI)**.
    5) MISO pin connected to **pin 50 (SPI MISO)**.
    6) 3.3V pin connected to the **3.3V** output of the Arduino.
    7) GND pin connected to **GND**.

  ii) Servo Motor:
    1) Orange (signal) wire connected to **digital pin 7**.
    2) Red (VCC) wire connected to **5V** on the Arduino.
    3) Brown (GND) wire connected to **GND**.

  iii) LED Indicators:
    1) Green LED anode was connected to **digital pin 5**, cathode connected to **GND**.
    2) Red LED anode was connected to **digital pin 6**, cathode connected to **GND**.

  iv) MPU6050 Sensor:
    1) VCC pin connected to **5V** on the Arduino.
    2) GND pin connected to **GND**.
    3) SDA pin connected to **pin 20 (SDA)** on the Arduino Mega.
    4) SCL pin connected to **pin 21 (SCL)** on the Arduino Mega.

  v) Arduino and Laptop Connection:
    1) The Arduino Mega board was connected to the **laptop via USB cable**.
    2) This connection provides **power supply** and allows **serial communication** between the Arduino and Python for gesture control and system commands.

b) Programming Logic

Python

i) Import Modules:
   1) **serial** was used to enable serial communication between Python and Arduino.
   2) **time** was used to handle timing, delays, and timeout control.

ii) Configuration and Initialization:
   1) **SERIAL_PORT = 'COM5'** was used to define Arduino's communication port.
   2) **BAUD_RATE = 9600** will set the data transfer rate.
   3) **AUTHORIZED_UIDS = ["4305CAF7"]** will store valid RFID tag UIDs.
   4) **GESTURE_THRESHOLD = 1.5** was the Minimum gyro Z-axis value for detecting a gesture.
   5) **GESTURE_TIMEOUT = 5.0** was the time limit (in seconds) for performing the gesture.
   6) **UNLOCK_DURATION = 5.0** was the time period for which the servo remained unlocked.
   7) Serial connection was opened by using serial.Serial() and needed to wait for 2 seconds for Arduino initialization.

iii) State Machine Setup:
   1) STATE_WAITING_FOR_UID(System waits for RFID tag scan).
   2) STATE_WAITING_FOR_GESTURE(Waits for user's gesture after valid RFID).
   3) STATE_UNLOCKED(Lock is open for limited duration).

iv) Main Loop: Continuous Monitoring
   1) Timeout Handling:
      - If gesture time exceeds 5s → send 'D' (Access Denied) and reset to UID state.
      - If unlock time exceeds 5s → send 'D' to relock and reset state.
   2) Serial Data Reading:
      - Continuously read incoming serial data from Arduino.
      - If the message starts with "UID:"→ extract UID and check authorization.
      - If the message starts with "MPU:" → extract motion data (gyro/accel).
   3) Decision Logic:
      - When waiting for UID:
         (i) If UID matches **AUTHORIZED_UIDS**, send 'G' to enter gesture mode.

(ii) Else, send ‘D’ to deny access.

- When waiting for gesture:
  (i) Parse gyro Z-axis value from MPU data.
  (ii) If gyro_z > 1.5 (threshold) → gesture detected.
  (iii) Send ‘A’ (Access Granted), unlock servo, and switch to unlocked state.
- When unlocked:
  (i) Wait until unlock duration expires, then relock automatically

v) Exit Handling:

1) On **KeyboardInterrupt (Ctrl+C)** or Arduino disconnection:
   - Send ‘D’ to ensure the lock is closed.
   - Close the serial connection safely.
   - Display message: “Serial port closed.”

Arduino

i) Include Libraries:

1) **SPI.h & MFRC522.h** was used to enable RFID reader communication (RC522).
2) **Servo.h** was used to control the servo lock mechanism.
3) **Wire.h, Adafruit_MPU6050.h, Adafruit_Sensor.h** was used to handle MPU6050 sensor readings for gesture detection.

ii) Pin Setup & Object Creation:

1) RFID: RST=9, SS=10 (SPI).
2) Servo: Pin 7.
3) LEDs: Green → Pin 5, Red → Pin 6.
4) Objects which belong for each rfid,lockServo and mpu were created for each module.

iii) Initialization (setup):

1) Start Serial at **9600 baud** (sync with Python).
2) Initialize **SPI** for RFID and **I2C** for MPU6050.
3) Set LED pins as outputs and start in a locked state (servo closed, red LED on).
4) Flush serial buffer and print “Arduino is ready”

iv) Main Loop Overview:

1) Handle Serial Commands (from Python):
   - ‘A’= Unlock (green LED on, servo opens).
   - ‘D’=Lock (red LED on, servo closes).
   - ‘G’ =Enter gesture mode (blink green LED, start sending MPU data).
2) RFID Scanning (when not in gesture mode):
   - Continuously check for RFID tag presence.

            - If found, send UID to Python via Serial.
        3) MPU6050 Data Transmission (in gesture mode):
            - Blink green LED every 300 ms.
            - Every 100 ms, read accelerometer & gyroscope data.
            - Send all 6 axes as: ax, ay, az, gx, gy, gz.
    v) Helper Functions:
        1) **setLockedState()** =Red LED ON, servo closes, RFID reinitialized.
        2) **setUnlockedState()** = Green LED ON, servo opens, then detaches
c) Code Used:
    i) Python

```python
import serial

import time


# --- Configuration ---

# !! CHANGE THIS to your Arduino's COM port

SERIAL_PORT = 'COM5'

BAUD_RATE = 9600

AUTHORIZED_UIDS = ["4305CAF7"]  # Your authorized token

GESTURE_THRESHOLD = 1.5  # Gyro Z-axis threshold (rad/s)

GESTURE_TIMEOUT = 5.0  # 5 seconds to perform the gesture

UNLOCK_DURATION = 5.0  # 5 seconds to keep the lock open


# --- State Machine Variables ---

STATE_WAITING_FOR_UID = "STATE_WAITING_FOR_UID"

STATE_WAITING_FOR_GESTURE = "STATE_WAITING_FOR_GESTURE"

STATE_UNLOCKED = "STATE_UNLOCKED"

currentState = STATE_WAITING_FOR_UID


gesture_timeout_start = 0.0

unlock_start_time = 0.0
```

```
def main():
    global currentState, gesture_timeout_start, unlock_start_time

    print("Connecting to Arduino...")
    try:
        ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
        time.sleep(2)  # Wait for serial connection

        # Read Arduino's startup message
        startup_message = ser.readline().decode('utf-8').strip()
        print(f"Arduino says: {startup_message}")

        print("System is armed. Waiting for RFID token...")

    except serial.SerialException as e:
         print(f"Error: Could not open serial port {SERIAL_PORT}.")
        print(f"Details: {e}")
        print("Please check the port and ensure Arduino IDE's Serial Monitor
is closed.")
        return

    try:
        while True:
                # --- 1. CHECK FOR TIMEOUTS (applies every loop) ---

                # Check for gesture timeout
```

```
            if currentState == STATE_WAITING_FOR_GESTURE and (time.time() -
gesture_timeout_start > GESTURE_TIMEOUT):
            print("Gesture timeout. ACCESS DENIED.")
            ser.write(b'D')  # Send 'D' for Deny
            currentState = STATE_WAITING_FOR_UID  # Reset state
            print("\nSystem re-armed. Waiting for RFID token...")


            # Check for unlock duration timeout
            if currentState == STATE_UNLOCKED and (time.time() -
unlock_start_time > UNLOCK_DURATION):
            print(f"Lock open for {UNLOCK_DURATION}s. Relocking...")
            ser.write(b'D')  # Send 'D' to relock
            currentState = STATE_WAITING_FOR_UID  # Reset state
            print("\nSystem re-armed. Waiting for RFID token...")


            # --- 2. READ DATA FROM ARDUINO ---
            try:
            line = ser.readline().decode('utf-8').strip()
            except UnicodeDecodeError:
            continue  # Ignore bad serial data


            # --- 3. PROCESS DATA (only if we received any) ---
            if line:


            # --- State 1: Waiting for UID ---
            if currentState == STATE_WAITING_FOR_UID:
                  if line.startswith("UID:"):
                        uid = line.split(":")[1].upper()
                        print(f"Token Scanned: {uid}")
                        if uid in AUTHORIZED_UIDS:
                          print(f"TOKEN AUTHORIZED. Perform circular gesture within {GESTURE_TIMEOUT} seconds...")
                          ser.write(b'G')  # Send 'G' to request gesture data

                          currentState = STATE_WAITING_FOR_GESTURE
                         gesture_timeout_start = time.time()
                        else:
                          print("ACCESS DENIED. Unknown token.")
                          ser.write(b'D')


            # --- State 2: Waiting for Gesture ---
            elif currentState == STATE_WAITING_FOR_GESTURE:
                  if line.startswith("MPU:"):
                        try:
                          parts = line.split(":")[1].split(',')
                          if len(parts) == 6:
                              gyro_z = float(parts[5])


                              # Print debug score
                              if abs(gyro_z) > 0.5:
                                    print(f"  ... (Gyro Z: {gyro_z:.2f})")


                              # Check for the gesture
                              if abs(gyro_z) > GESTURE_THRESHOLD:
                                  print(f"GESTURE DETECTED! (Gyro Z: {gyro_z:.2f})")
                                    print(f"ACCESS GRANTED. Unlocking for {UNLOCK_DURATION} seconds...")
```

```
                                     ser.write(b'A')  # Send 'A' for Access

                                     currentState = STATE_UNLOCKED

                                   unlock_start_time = time.time()


                         except (ValueError, IndexError):

                           pass  # Ignore malformed MPU data


            # --- State 3: Unlocked ---

            # The timeout check at the top handles relocking.

            elif currentState == STATE_UNLOCKED:

                  pass


   except KeyboardInterrupt:

      print("\nProgram terminated by user.")

   except serial.SerialException:

      print("\nError: Arduino disconnected.")

   finally:

      if 'ser' in locals() and ser.is_open:

            ser.write(b'D')  # Ensure it's locked before closing

            ser.close()

            print("Serial port closed.")



if __name__ == "__main__":

   main()
```

ii) Arduino

```
  This sketch is for Task 2 (Python-Controlled System).

  It has been updated to only send MPU data when requested by Python.

  It listens for:

  - 'A': Access (Unlock servo, green LED)

  - 'D': Deny (Lock servo, red LED)

  - 'G': Gesture (Start sending MPU data, blink green LED)

*/



// --- Include Libraries ---

#include <SPI.h>

#include <MFRC522.h>

#include <Servo.h>

#include <Wire.h>

#include <Adafruit_MPU6050.h>

#include <Adafruit_Sensor.h>



// --- Pin Definitions (for your MEGA) ---

#define RFID_RST_PIN 9

#define RFID_SS_PIN  10 // SPI SS (The "SDA" pin on the RC522)

#define SERVO_PIN7

#define LED_GREEN_PIN 5

#define LED_RED_PIN   6

// MPU6050 uses dedicated I2C pins: 20 (SDA) and 21 (SCL)
// --- Create Objects ---

MFRC522 rfid(RFID_SS_PIN, RFID_RST_PIN);

Servo lockServo;

Adafruit_MPU6050 mpu;


// --- Global Variables ---

unsigned long lastMpuReadTime = 0;

const long mpuReadInterval = 100; // Read MPU every 100ms

const int OPEN_ANGLE = 180;

const int CLOSED_ANGLE = 0;


bool sendMpuData = false; // Flag to control MPU data streaming

unsigned long lastBlinkTime = 0;

bool greenLedState = false;


void setup() {

  Serial.begin(9600); // Must match Python's baud rate


  // Init SPI (for RFID)

  SPI.begin();

  rfid.PCD_Init();


  // Init I2C (for MPU6050)

  Wire.begin();

  if (!mpu.begin()) {
    Serial.println("MPU6050 connection failed! Check wiring.");

    while (1); // Stop here

  }



  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);

  mpu.setGyroRange(MPU6050_RANGE_500_DEG);

  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);



  // Init LEDs

  pinMode(LED_GREEN_PIN, OUTPUT);

  pinMode(LED_RED_PIN, OUTPUT);



  // Set initial LOCKED state

  setLockedState();



  // --- CRITICAL FIX ---

  // Flush the serial buffer

  while(Serial.available()) Serial.read();



  Serial.println("Arduino is ready. Waiting for Python commands.");

}



void loop() {

  // --- Task 1: Check for Serial Commands FROM Python ---

  if (Serial.available() > 0) {
      char command = Serial.read();


      if (command == 'A') { // Access Granted

      setUnlockedState();

      sendMpuData = false; // Stop sending MPU data


      } else if (command == 'D') { // Access Denied or Relock

      setLockedState();

      sendMpuData = false; // Stop sending MPU data


      } else if (command == 'G') { // Gesture mode

      sendMpuData = true; // Start sending MPU data

      lastBlinkTime = millis(); // Start blinking

      greenLedState = true;

      digitalWrite(LED_GREEN_PIN, greenLedState);

      digitalWrite(LED_RED_PIN, LOW); // Turn red off

      }

  }


  // --- Task 2: Check for RFID Tag (Non-blocking) ---

  // Only scan for RFID if we are not in gesture mode

  if (!sendMpuData) {

      if (rfid.PICC_IsNewCardPresent() && rfid.PICC_ReadCardSerial()) {

      Serial.print("UID:"); // Send UID to Python

      for (byte i = 0; i < rfid.uid.size; i++) {
```

```
    Serial.print(rfid.uid.uidByte[i] < 0x10 ? "0" : "");

    Serial.print(rfid.uid.uidByte[i], HEX);

    }

    Serial.println();


    rfid.PICC_HaltA();

    rfid.PCD_StopCrypto1();

    }

}


// --- Task 3: Read MPU6050 (if requested) ---

if (sendMpuData) {

    // Handle blinking green LED

    if (millis() - lastBlinkTime > 300) {

    lastBlinkTime = millis();

    greenLedState = !greenLedState;

    digitalWrite(LED_GREEN_PIN, greenLedState);

    }


    // Handle sending MPU data

    if (millis() - lastMpuReadTime > mpuReadInterval) {

    lastMpuReadTime = millis();

    sensors_event_t a, g, temp;

    mpu.getEvent(&a, &g, &temp);
```

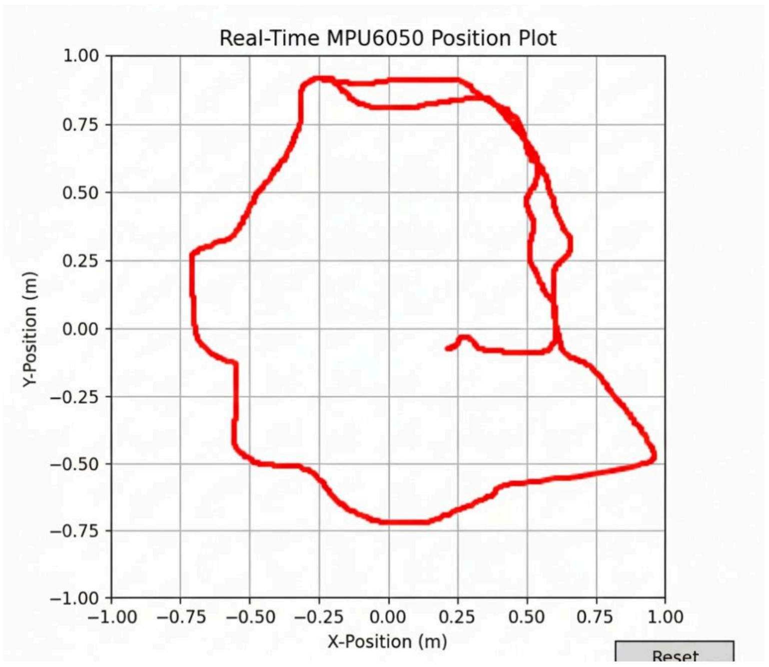
```
      // Send all 6 axes to Python

      Serial.print("MPU:");

      Serial.print(a.acceleration.x); Serial.print(",");

      Serial.print(a.acceleration.y); Serial.print(",");

      Serial.print(a.acceleration.z); Serial.print(",");

      Serial.print(g.gyro.x); Serial.print(",");

      Serial.print(g.gyro.y); Serial.print(",");

      Serial.print(g.gyro.z);

      Serial.println();

      }

  }

}



// --- Helper Functions ---



void setLockedState() {

  digitalWrite(LED_GREEN_PIN, LOW);

  digitalWrite(LED_RED_PIN, HIGH);

  lockServo.attach(SERVO_PIN);

  lockServo.write(CLOSED_ANGLE);

  delay(500); // Give servo time to move

  lockServo.detach();

  rfid.PCD_Init(); // Re-init RFID reader

}
```

```
void setUnlockedState() {

  digitalWrite(LED_GREEN_PIN, HIGH); // Solid Green

  digitalWrite(LED_RED_PIN, LOW);

  lockServo.attach(SERVO_PIN);

  lockServo.write(OPEN_ANGLE);

  delay(500); // Give servo time to move

  lockServo.detach();

}
```

5.0 Data Collection

5.1 Hand Gesture System

Motion data were collected from the MPU6050 sensor in real time through the Arduino serial interface. The data included raw accelerometer and gyroscope readings corresponding to various hand movements. The information was then visualized using Python, where x-y acceleration data were plotted dynamically to illustrate the motion path and verify gesture recognition accuracy.



5.2 RFID System

The RFID module was integrated with the servo motor and LED indicators to implement an access-control system. The Arduino continuously scanned for RFID tags using the MFRC522 reader.

When a tag was detected, the module transmitted its unique identifier (UID) through the serial connection to the Python program. The Python script compared the received UID against a predefined list of authorized IDs stored directly in the source code. If the scanned UID matched an authorized entry, the system granted access by transmitting an ‘A’ command to the Arduino. Otherwise, it sent a ‘D’ command to indicate denial.

Upon receiving ‘A,’ the Arduino rotated the servo motor to the open position (180°) and illuminated the green LED. When ‘D’ was received , the servo returned to the locked position (0°), and the red LED turned on. The system automatically relocked after a fixed delay or when an unauthorized tag was scanned.

6.0 Data Analysis

6.1 Hand Gesture System

The raw data collected from the MPU6050 sensor, comprising linear acceleration and angular velocity along the X, Y, and Z axes, were analyzed to interpret hand movements. The Z-axis reading remained relatively constant, indicating a stable upright orientation. The X and Y values fluctuated within a range that corresponded to lateral and forward hand motions, confirming the successful detection of simple directional gestures. Gyroscope data further supported the identification of rotational motions

6.2 RFID System

The collected data showed that the system correctly distinguished between authorized and unauthorized RFID tags. When an authorized UID was scanned, the servo motor rotated smoothly to the unlock position, and the green LED illuminated to signal access approval.

After the set duration, the servo automatically returned to its initial locked position, confirming proper timed control. For all other tags, the system denied access immediately, turning on the red LED without any servo movement.

Serial communication between Python and Arduino remained stable, and all feedback responses were consistent with expected system behavior.

7.0 Result

7.1 Hand Gesture System

The MPU6050 successfully transmitted accelerometer and gyroscope data through the I2C interface, allowing real-time plotting in Python. The Arduino program correctly recognized predefined gestures by evaluating acceleration thresholds. When gestures were performed, they were detected and displayed via serial communication, confirming accurate classification.

7.2 RFID System

The RFID access control successfully validated tag identification and executed correct mechanical and visual responses. The servo motor and LEDs reacted in real time to Python commands, confirming seamless serial communication between the computer and the microcontroller.

Access for the authorized UID “4305CAF7” was granted reliably, and all unlisted UIDs were rejected. The automatic relocking function operated accurately, ensuring secure operation without user intervention.

Overall, the system performed as intended, integrating RFID sensing, logic control, and actuator feedback into a cohesive setup.

8.0 Discussion

The experiment successfully demonstrated the integration of RFID technology with motion sensing to create a responsive and secure access-control system. The implementation utilized direct communication between Python and Arduino, allowing the computer to handle logic processing while the microcontroller managed actuator responses. This setup effectively showed the use of serial interfacing for real-time decision-making and control in an embedded environment.

In this project, the RFID system operated by comparing scanned tag UIDs against a predefined list stored within the Python script rather than an external database. This approach simplified program execution and reduced data dependency, though it required manual modification whenever new users were added. The servo motor and LED indicators provided immediate physical feedback, which verified whether access was granted or denied.

The inclusion of gesture-based motion detection using the MPU6050 sensor added an additional layer of authentication. The Python program analyzed gyroscope data. When the motion exceeded the predefined threshold, the system confirmed a calid gesture an granted access. This dual-layer mechanism combined both RFUD identification and physical gesture verification, improving the reliability and uniqueness of the access control process.

However, several limitations were identified. The fixed threshold for gesture detection might not adapt well to varying user speeds or motion intensities. Sensor noise and environmental vibration could also affect the accuracy of the MPU6050 readings. IN addition, since the list of authorized UIDs was embedded directly in Python code, scalability was limited. To add or remove users, the script required manual editing. Furthermore, serial communication timing had to be managed carefully to ensure smooth coordination between Python and Arduino without data collision or delay.

9.0 Conclusion

The overall system functioned as expected. The RFID scanning, gesture detection, and actuator were consistently reliable during testing. The MPU6050 sensor accurately detected rotational gestures, proving effective for motion-based authentication. The integration between software-based decision logic and hardware execution demonstrated a practical model of a two-step authentication system, which could be further refined for real-world security or automation applications.

10. Recommendations

It was recommended to replace the hard-coded UID list with an external storage method such as a JSON or CSV file, enabling easier addition or removal of authorized users without modifying source code to improve scalability.

Implementing encrypted UID transmission could further enhance system security and prevent impersonation. A real-time logging feature in the Python program would assist in tracking entry events for monitoring purposes.

In addition, incorporating a graphical user interface (GUI) would make the system more user-friendly, allowing operators to view detected tags, update authorized IDs, and control the locking mechanism visually.

Mechanism improvement, like adding limit switches or position feedback for the servo, would ensure consistent locking accuracy during long-term use.

11.0 References

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

MPU6050 6-DoF Accelerometer and Gyro. (n.d.). Adafruit Learning System. https://learn.adafruit.com/mpu6050-6-dof-accelerometer-and-gyro/arduino

12.0 Acknowledgement

We would like to express our sincere gratitude to Dr. Wahju Sediono, Dr. Zulkifli Bin Zainal Abidin, our teaching assistant, and our peers for their invaluable guidance and support throughout the completion of this report. Their insights, feedback, and expertise greatly contributed to the depth and quality of this work. We truly appreciate their time, patience, and commitment to my academic growth.

13.0 Student's Declaration

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 no further improvement on the reports is needed from any of the individual's contributors to the report.

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

Name: Shamsul Hakimee Bin Shamsul Hairee	Read	[/]
Matric Number: 2315027	Understand	[/]
Signature: [illegible]	Agree	[/]

Name: Qashrina Aisya binti Mohd Nazarudin	Read	[/]
Matric Number: 2315184	Understand	[/]
Signature: [illegible]	Agree	[/]

Name: Nurel Humairah Binti Muhammad Firdaus	Read	[/]
Matric Number: 2315680	Understand	[/]
Signature: [illegible]	Agree	[/]