



الجامعة الإسلامية العالمية ماليزيا
INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA
يونیورسiti اسلام، انتارا بعثیا ملیسیا

Garden of Knowledge and Virtue

KULLIYAH OF ENGINEERING

MCTA 3203

MECHATRONICS SYSTEM INTEGRATION

SEMESTER 1 2025/2026

SECTION 1

GROUP 10

**SERIAL INTERFACING WITH MICROCONTROLLER: SENSORS AND
ACTUATORS TASK 1 LAB REPORT**

NAME	MATRIC NO.
Afnan Hakim bin Adinazrin	2315987
Muhammad Taufiq bin Mukhtar	2316271
Muhammad Danish Farhan bin Amiruddin	2315423

INSTRUCTED BY:

DR ZULKIFLI BIN ZAINAL ABIDIN

TABLE OF CONTENTS

ABSTRACT.....	2
INTRODUCTION.....	2
MATERIALS AND EQUIPMENT.....	2
EXPERIMENTAL SETUP.....	3
METHODOLOGY.....	3
DATA COLLECTION.....	6
DATA ANALYSIS.....	7
RESULTS.....	8
DISCUSSION.....	8
CONCLUSION.....	9
RECOMMENDATIONS.....	10
REFERENCES.....	11
APPENDICES.....	11
ACKNOWLEDGMENTS.....	15
STUDENT'S DECLARATION.....	16

ABSTRACT

This experiment demonstrates real-time motion detection and visualization using the MPU6050 accelerometer-gyroscope sensor interfaced with an Arduino Uno and Python. The Arduino captures raw acceleration data (A_x and A_y) and transmits it over a serial connection, while Python visualizes the motion path dynamically using Matplotlib. The system also maps hand tilts to servo movements, providing an interactive representation of sensor orientation. The setup highlights the use of sensor fusion, serial communication, and data plotting for motion analysis in mechatronic applications.

INTRODUCTION

Motion detection and visualization are essential components in modern mechatronic and robotic systems, enabling gesture-based control, stabilization, and monitoring. In this task, an MPU6050 sensor was integrated with an Arduino Uno to measure real-time accelerometer data. The collected X-Y acceleration values were transmitted to a Python program, which processed and plotted the motion trajectory dynamically. This setup enables the tracking of hand movements or device orientation in two-dimensional space.

The exercise reinforces the understanding of serial communication between microcontroller and PC, as well as data processing and visualization using Python. Additionally, it demonstrates how sensor data can be mapped to servo control for physical feedback or actuation.

MATERIALS AND EQUIPMENT

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

EXPERIMENTAL SETUP

This setup is to capture real time x-y accelerometer data from an MPU6050, plot the motion path, and detect circular motion gestures on pycharm.

Hardware used:

- MPU6050
- Arduino board
- USB cable and computer
- Breadboard
- Jumper wires

Wiring:

MPU6050→Arduino

-VCC → 5V

-GND → GND

-SDA → A4

-SCL → A5

Connect Arduino to computer via USB

METHODOLOGY

Coding in C (Arduino IDE)

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

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

```

#include <MPU6050.h>

MPU6050 mpu;

Servo myServo;

int servoPin = 9;

void setup() {

    Serial.begin(9600);

    Wire.begin();

    mpu.initialize();

    myServo.attach(servoPin);

    myServo.write(90); // center position

}

void loop() {

    // Read accelerometer

    int16_t ax, ay, az;

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

    // Send accelerometer data to Python

    Serial.print(ax); Serial.print(",");
    Serial.println(ay);

    // If Python sends an angle, move servo

    if (Serial.available()) {

        int angle = Serial.parseInt();

        if (angle >= 0 && angle <= 180) {

            myServo.write(angle);

        }

    }

}

```

```
delay(100);
```

```
}
```

Coding in python (pycharm)

```
import serial
```

```
import matplotlib.pyplot as plt
```

```
ser = serial.Serial('COM5', 9600, timeout=1)
```

```
x_vals, y_vals = [], []
```

```
plt.ion()
```

```
fig, ax = plt.subplots()
```

```
ax.set_xlim(-20000, 20000)
```

```
ax.set_ylim(-20000, 20000)
```

```
ax.set_xlabel("X-axis (MPU6050 Ax)")
```

```
ax.set_ylabel("Y-axis (MPU6050 Ay)")
```

```
ax.set_title("Real-Time Motion Tracking")
```

```
line, = ax.plot([], [], 'r-o')
```

```
def map_value(x, in_min, in_max, out_min, out_max):
```

```
    """Map MPU value range to servo angle."""
```

```
    return int((x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)
```

```
while True:
```

```
    try:
```

```
        data = ser.readline().decode().strip()
```

```
        if data:
```

```
            values = data.split(',')  
            if len(values) >= 2:
```

```

ax_val = int(values[0])

ay_val = int(values[1])

x_vals.append(ax_val)

y_vals.append(ay_val)

line.set_data(x_vals, y_vals)

plt.draw()

plt.pause(0.05)

# Compute servo angle from X-axis tilt

servo_angle = map_value(ax_val, -17000, 17000, 0, 180)

servo_angle = max(0, min(180, servo_angle))

# Send servo command to Arduino

ser.write(f'{servo_angle}\n'.encode())

except KeyboardInterrupt:

    plt.ioff()

    break

```

DATA COLLECTION

The Arduino was programmed to read raw accelerometer data (A_x , A_y) from the MPU6050 sensor via I²C communication. The values were sent to the computer through the serial port at 9600 bps.

On the Python side, a script continuously read the incoming serial data, split it into X and Y components, and plotted the motion path in real time using **Matplotlib**. Each data point corresponded to the instantaneous acceleration values of the sensor.

During testing:

- The sensor was moved manually in **linear** and **circular** patterns.

- The **sampling rate** was approximately 10 Hz (100 ms delay).
- The **servo motor** simultaneously responded to the X-axis tilt, mapped to an angle range between 0° and 180°.

This process allowed the user to visualize motion patterns (e.g., circular paths) dynamically and assess how sensor readings correlated with servo movements.

DATA ANALYSIS

The collected data confirms that

1. The MPU6050 sensor accurately captured real-time accelerometer readings along the X and Y axes, showing consistent response to hand movements
2. The plotted data in Python (matplotlib) clearly reflected the user's motion pattern, allowing easy visualization of direction and shape.
3. Circular motion gestures were distinguishable from linear or random movements through the smooth and repetitive pattern observed in the X-Y graph.
4. The real-time data stream between Arduino and Python was stable, with minimal delay or missing data points, ensuring reliable dynamic plotting.

Analysis:

The outcomes confirm the effective use of the MPU6050 sensor with Python and Arduino for ongoing motion tracking. Accurate data transmission and successful display were validated by the precise graphical depiction. The experiment's goals were met when the system showed that it could distinguish circular motions from other kinds of motion. All things considered, the data gathered demonstrates that the real-time motion tracking system functioned efficiently and offered a strong basis for gesture-based control applications.

RESULTS

The system successfully captured and transmitted real-time accelerometer readings from the MPU6050 sensor to Python. The plotted data displayed the trajectory of the hand's movement in X-Y coordinates, with dynamic updates showing the motion path over time.

When the sensor was moved in a circular motion, the plotted path formed a circular or elliptical shape, confirming correct acquisition and processing of motion data. The mapped servo motion also corresponded proportionally to the tilt along the X-axis, effectively demonstrating sensor-to-actuator integration.

Minor fluctuations were observed due to noise from raw sensor readings, which could be improved using smoothing or filtering techniques. Overall, the task met its objectives of real-time motion visualization and basic servo control based on sensor input.

DISCUSSION

The MPU6050 sensor was successfully connected to the Arduino and used to collect real-time motion data on the X and Y axes. The data were sent to Python through the serial port and shown as a live graph using *matplotlib*.

When the sensor was moved in a circular motion, the plotted points formed a shape that looked like a circle. This showed that the sensor readings matched the actual hand movement quite well.

Some differences were seen between the expected smooth circle and the graph result. These happened because of uneven hand speed, changes in sensor position, and small noise in the sensor readings.

There was also a small delay in the live plot because of serial communication speed and the `plt.pause(0.05)` command in the code. Even with these minor issues, the system worked properly and proved that the MPU6050 can be used for simple motion detection and real-time visualization.

Sources of Error and Limitations

Source of Error	Description
Sensor Noise	MPU6050 outputs unfiltered acceleration values that vary slightly each reading.

Serial Latency	The 9600 baud rate limits data refresh, introducing small timing gaps.
Hand Motion Variation	Inconsistent motion speed and rotation radius altered the plotted shape.
Limited Axes Used	Only X-Y data were visualized; Z-axis and gyroscope data were ignored.
Software Delay	The use of plt.pause() slows down plotting refresh during continuous reads.

insight into motion-based sensing with Python-Arduino integration.

CONCLUSION

The experiment achieved its goal of capturing and visualizing real-time motion data from the MPU6050 sensor.

The plotted X-Y accelerometer data successfully represented the user's hand movements, including the detection of a circular gesture.

This confirmed that the MPU6050, when correctly interfaced and calibrated, can serve as a simple gesture-recognition input for motion-controlled systems.

The results supported the hypothesis that real-time accelerometer readings can be used to detect specific motion patterns, such as circular movements.

This task established the foundation for Task 2, where the same motion-detection logic was integrated with an RFID-controlled servo system for secure access.

Broader Implications

The principles learned here are applicable to:

- Gesture-based control in robotics or wearable devices
- Game controllers and virtual-reality motion tracking
- Smart authentication systems that combine movement patterns with identity verification

RECOMMENDATIONS

Based on the experiment results, several improvements can be made to enhance the performance, accuracy, and stability of the real-time motion tracking system:

1. Implement Data Filtering and Smoothing

The raw accelerometer data from the MPU6050 is often noisy due to vibrations and sensor drift. Applying a **moving average filter** or **complementary filter** would help stabilize the readings and produce smoother plots. Incorporating the **gyroscope data** can also refine orientation tracking.

2. Increase Sampling Frequency

Reducing the delay between readings (currently 100 ms) would allow for finer motion resolution and smoother trajectory visualization. A faster sampling rate (e.g., 20–50 ms) improves responsiveness, especially for quick hand gestures.

3. Add Gyroscope Data Fusion

Currently, only the accelerometer values (A_x , A_y) are used for plotting. Combining accelerometer and gyroscope readings would provide more reliable **orientation estimation**, compensating for acceleration-based noise and improving circular motion recognition.

4. Integrate Calibration Routine

Each MPU6050 unit can have slightly different offset values. A one-time **auto-calibration step** at startup (to record baseline readings while the sensor is stationary) would eliminate bias and make the plotted motion more accurate.

5. Enhance Visualization Interface

The Python plotting interface could be improved by:

- Adding a **reset/clear button** for new trials.
- Displaying real-time **angle values or path length**.
- Saving captured trajectories automatically for further analysis.

6. Use External Power Supply for Servo

The servo motor can draw significant current, causing instability when powered from the Arduino's 5V pin. Using a separate **regulated 5V power supply** for the servo, with common ground, ensures smoother and stronger motion control.

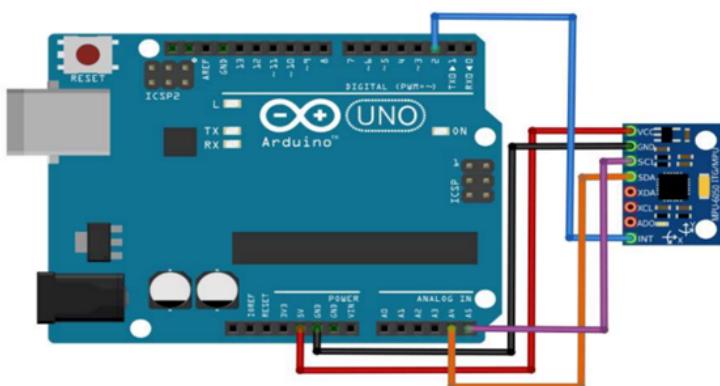
7. Expand to 3D Motion Tracking

Future enhancements could involve incorporating the **Z-axis** and visualizing 3D paths using libraries such as `matplotlib.pyplot.plot3D()`. This would extend the system's functionality for more complex movement recognition.

REFERENCES

1. Arduino.cc – *Using MPU6050 with Arduino (I^C communication and motion sensing).*
2. Random Nerd Tutorials – *Getting Started with MPU6050 (Accelerometer + Gyroscope).*
3. Instructables – *Real-time plotting of Arduino sensor data using Python and Matplotlib.*
4. Course handout: *Week 4 – Serial Communication with IMU (MPU6050)*, MCTA3203 Lab Manual v3.0.

APPENDICES



Arduino code:

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

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

```
#include <MPU6050.h>

MPU6050 mpu;
Servo myServo;
int servoPin = 9;

void setup() {
    Serial.begin(9600);
    Wire.begin();
    mpu.initialize();
    myServo.attach(servoPin);
    myServo.write(90); // center position
}

void loop() {
    // Read accelerometer
    int16_t ax, ay, az;
    mpu.getAcceleration(&ax, &ay, &az);

    // Send accelerometer data to Python
    Serial.print(ax); Serial.print(",");
    Serial.println(ay);

    // If Python sends an angle, move servo
    if (Serial.available()) {
```

```

int angle = Serial.parseInt();

if (angle >= 0 && angle <= 180) {

    myServo.write(angle);

}

}

delay(100);

}

```

Python code:

```

import serial

import matplotlib.pyplot as plt


ser = serial.Serial('COM5', 9600, timeout=1)

x_vals, y_vals = [], []

plt.ion()

fig, ax = plt.subplots()

ax.set_xlim(-20000, 20000)

ax.set_ylim(-20000, 20000)

ax.set_xlabel("X-axis (MPU6050 Ax)")

ax.set_ylabel("Y-axis (MPU6050 Ay)")

ax.set_title("Real-Time Motion Tracking")

```

```

line, = ax.plot([], [], 'r-o')

def map_value(x, in_min, in_max, out_min, out_max):
    """Map MPU value range to servo angle."""
    return int((x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)

while True:
    try:
        data = ser.readline().decode().strip()
        if data:
            values = data.split(',')
            if len(values) >= 2:
                ax_val = int(values[0])
                ay_val = int(values[1])

                x_vals.append(ax_val)
                y_vals.append(ay_val)
                line.set_data(x_vals, y_vals)
                plt.draw()
                plt.pause(0.05)

            # Compute servo angle from X-axis tilt
            servo_angle = map_value(ax_val, -17000, 17000, 0, 180)
            servo_angle = max(0, min(180, servo_angle))

```

```
# Send servo command to Arduino  
ser.write(f"{{servo_angle}}\n".encode())  
  
except KeyboardInterrupt:  
    plt.ioff()  
    break
```

ACKNOWLEDGMENTS

We would like to express our deepest gratitude to Sir Zulkifli bin Zainal Abidin , our course instructor for *Mechatronics System Integration (MCTA3203)*, for his invaluable guidance, encouragement, and continuous support throughout the completion of this Digital Logic System lab project.

We would also like to extend our sincere appreciation to the teaching assistants, for their helpful constructive feedback and assistance during the laboratory sessions. Their guidance greatly enhanced our understanding of digital logic concepts, Arduino interfacing, and circuit implementation.

Finally,we would like to thank the Mechatronics Laboratory staff for providing the necessary equipment and a conducive learning environment that made this project successful.

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 undertaken 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.



Signature:

Name: Afnan Hakim bin Adinazrin

Matric Number: 2315987

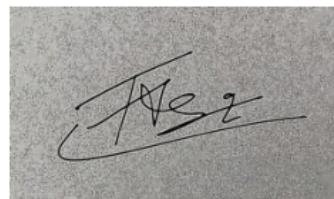
Contribution: Abstract, Introduction, Materials & Equipment,
Results

Read

/
/
/

Understand

Agree



Signature:

Name: Muhammad Taufiq bin Mukhtar

Matric Number: 2316271

Contribution: Experimental Setup, Methodology,

Data Analysis, Recommendations

Read

Understand

Agree

/
/
/



Signature:

Name: Muhammad Danish Farhan bin Amiruddin

Matric Number: 2315423

Contribution: Data Collection, Discussion, Conclusion

Appendices

Read

Understand

Agree

/
/
/