REPORT ON MECHATRONICS SYSTEM INTEGRATION

4A- SERIAL COMMUNICATION IMU
GROUP 7

SECTION 2, SEMESTER 1, 24/25

TEAM MEMBERS

NAME	MATRIC NO.
MUHAMMAD ZAMIR FIKRI BIN MOHD ZAMRI	2212515
MUHD AKMAL HAKIM BIN SAIFUDDIN	2216093
NUR SHADATUL BALQISH BINTI SAHRUNIZAM	2212064
NORHEZRY HAKIMIE BIN NOOR FAHMY	2110061
NUR AMIRA NAZIRA BINTI MOHD NASIR	2110026

DATE OF EXPERIMENT: 30 OCTOBER 2024
DATE OF SUBMISSION: 5 NOVEMBER 2024

ABSTRACT

This report presents the design and implementation of a gesture recognition system using the MPU6050 Inertial Measurement Unit (IMU) and an Arduino microcontroller, with data processed and visualized through Python. The system captures accelerometer and gyroscope data to detect and classify specific hand gestures, demonstrating real-time monitoring through serial communication. This experiment aims to explore the practical applications of IMU sensors for gesture-based controls in mechatronic systems, which are essential for human-computer interaction and industrial automation.

The project utilizes a threshold-based method to classify gestures, where different hand movements are detected based on accelerometer values along the x and y axes. Python code interprets the sensor data transmitted from the Arduino and displays detected gestures, providing immediate feedback to users. This report also discusses the challenges encountered, including sensor noise and calibration needs, and proposes recommendations for further optimizing accuracy. Overall, the experiment successfully validates the capability of IMU-based systems to perform reliable gesture detection, underscoring the potential of these technologies in areas such as robotics and wearable devices.

TABLE OF CONTENT

INTRODUCTION

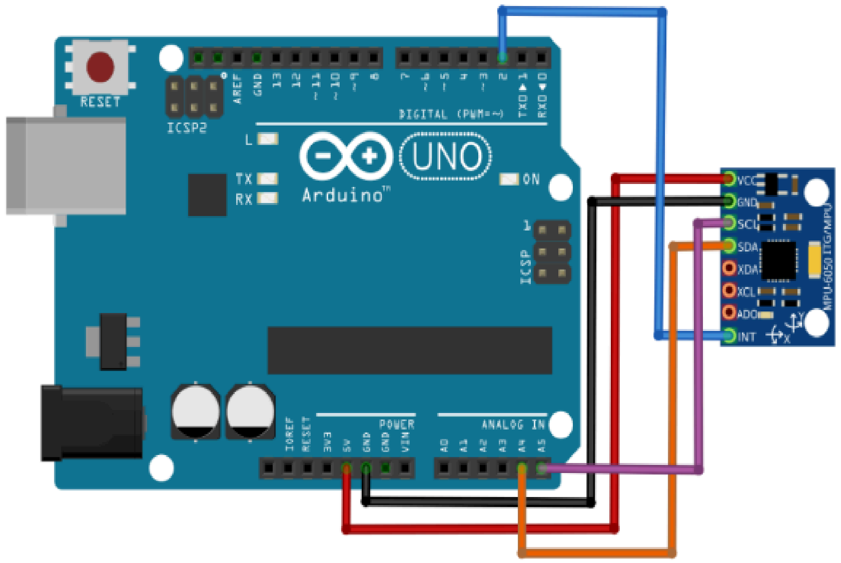
This project explores the integration of the MPU6050 Inertial Measurement Unit (IMU) sensor with an Arduino microcontroller to achieve real-time gesture recognition through serial communication. The MPU6050 sensor combines accelerometer and gyroscope functionalities, which allows it to capture movement data across multiple axes, enabling accurate gesture identification. This experiment focuses on building a simple yet effective mechatronic system where hand gestures are detected and classified using Python and Arduino-based software.

The objectives of this experiment are to analyze the accelerometer and gyroscope data to detect specific hand movements and to visualize these gestures through Python-based serial communication. The data collected by the MPU6050 is transmitted to a Python interface via Arduino, allowing for real-time monitoring and response to pre-defined gestures. Through this setup, the project demonstrates the practical applications of sensors in human-computer interaction and industrial monitoring.

In addition to gesture recognition, this project also aims to address challenges related to data noise, sensor calibration, and communication delays, enhancing the reliability and precision of IMU-based applications. The experiment not only provides insights into the capabilities of IMU sensors in embedded systems but also highlights the potential of these systems in fields ranging from robotics to wearable technology.

MATERIALS AND EQUIPMENT

1. Arduino Board
2. MPU6050 sensor
3. Computer with Arduino IDE and Python installed
4. Connecting wires
5. USB cable
6. Power Supply
7. LED lights



EXPERIMENTAL SETUP

1. Use the proper pins to connect the MPU6050 sensor to the Arduino board. Since the MPU6050 normally communicates via I2C, connect its SDA and SCL pins to the appropriate Arduino pins (generally A4 and A5 for the majority of Arduino boards).
2. Attach the MPU6050's ground and power supply to the 5V and GND pins on the Arduino.
3. Verify the Arduino board's USB connection to your computer.

METHODOLOGY

Step 1: Setting Up the Arduino with MPU6050

1. Connect the MPU6050: The MPU6050 accelerometer and gyroscope sensor is connected to the Arduino via I2C (using the SDA and SCL pins). The Wire library is included for I2C communication.
2. Initialize the MPU6050: In the setup() function, the MPU6050 is initialized, and a connection test is performed. If the test is successful, "MPU6050 connection successful" is printed to the Serial Monitor. If it fails, "MPU6050 connection failed" is displayed.
3. Set Up Serial Communication: The Serial communication is initialized at 9600 baud to enable data transfer between the Arduino and the computer/Python program.

Step 2: Detecting Gestures

1. Reading Sensor Data: In each loop, the detectGesture() function reads acceleration and gyroscope values (ax, ay, az, gx, gy, gz) from the MPU6050 sensor.
2. Defining Gestures: Based on specific conditions, gestures are detected:
    - If the ax value exceeds the threshold and ay is below the threshold, it recognizes Gesture 1.
    - If the ax is less than -threshold and ay is above the threshold, it recognizes Gesture 2.
    - Additional gestures can be added by specifying other conditions.
3. Outputting Detected Gestures: If a gesture is detected and differs from the previously detected gesture (previousGesture), it is printed to the Serial Monitor (e.g., "Detected Gesture: Gesture 1").

Step 3: Python Code for Serial Communication

1. Setting Up Serial Communication: A serial connection is established with the Arduino on COM4 (change as needed), matching the 9600 baud rate.
2. Reading Data from Arduino: A continuous loop reads the serial data from the Arduino, decoding and stripping any extra whitespace.
3. Handling Gesture Commands: When the received data indicates a detected gesture (e.g., "Detected Gesture: Gesture 1"):
    - The data is parsed to identify the specific gesture.
    - An action specific to that gesture is executed and printed in Python (e.g., "Action for Gesture 1").
4. Program Interruption: The program runs indefinitely, but if manually interrupted (e.g., KeyboardInterrupt), the serial connection is properly closed.

Step 4: Testing and Calibration

1. Run the Arduino Code: Upload the Arduino code and open the Serial Monitor to verify the sensor connection and ensure gesture detection outputs correctly.
2. Adjust Thresholds: Adjust the threshold variable in the Arduino code if gestures are not recognized accurately. Fine-tuning may be needed based on movement strength or noise in sensor readings.
3. Run the Python Program: Start the Python script and observe responses to gestures detected by the Arduino. Adjust actions in Python as necessary for different gestures.

Arduino Code:

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

MPU6050 mpu;

const int threshold = 1000; // Adjust this threshold as needed
int previousGesture = -1;

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

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

else {
Serial.println("MPU6050 connection successful");
}

}

void loop() {
int gesture = detectGesture();

if (gesture != previousGesture) {
Serial.print("Detected Gesture: ");

if (gesture == 1) {
```

```
Serial.println("Gesture 1");
// Perform an action for Gesture 1
}

else if (gesture == 2) {
Serial.println("Gesture 2");
// Perform an action for Gesture 2
}

// Add more gesture cases as needed
previousGesture = gesture;
}
delay(100);
}

int detectGesture() {

int ax, ay, az, gx, gy, gz;

mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
// Perform gesture recognition here based on sensor data
// Define conditions to recognize specific gestures
if (ax > threshold && ay < threshold) {
return 1; // Gesture 1
}

else if (ax < -threshold && ay > threshold) {
return 2; // Gesture 2
}

// Add more gesture conditions as needed
return 0; // No gesture detected
}
```

Python Code:

```
import serial
```

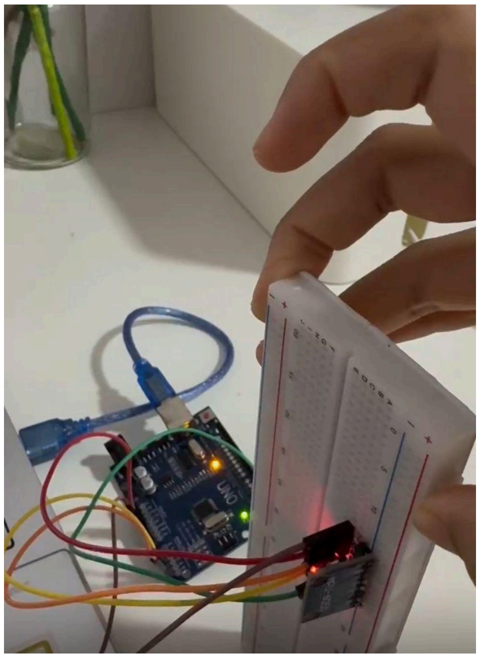
```
ser = serial.Serial('COM4', 9600)

try:
  while True:
    data = ser.readline().decode('utf-8').strip()
    if data:
            print(f"Received data: {data}")
            # Added feedback
    if data.startswith("Detected Gesture: "):
        gesture = data.split(": ")[1]
        if gesture == "Gesture 1":
            # Perform an action for Gesture 1
            print("Action for Gesture 1")
        elif gesture == "Gesture 2":
            # Perform an action for Gesture 2
            print("Action for Gesture 2")
        # Add more gesture actions as needed

except KeyboardInterrupt:
    print("Program stopped by user.")
finally:
    ser.close()  # Ensure the serial connection is closed when done
```

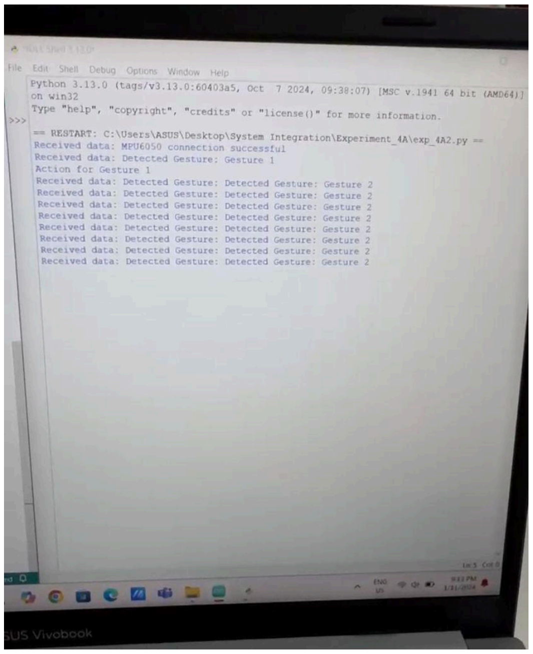
DATA ANALYSIS

Using the accelerometer and gyroscope data from the MPU6050, the analysis for this experiment focused on hand gesture detection. Applying a threshold-based classification method, two specified gestures were identified by applying particular numerical necessities to the x and y acceleration values. Gesture 1 had a lower value for ay (the y-axis acceleration) and a positive threshold for axe (the x-axis acceleration). On the other hand, a positive threshold for ay and a negative threshold for axe were used to identify Gesture 2. The threshold-based method allowed the system to categorize different gestures based on hand motions that were directed and transformed well onto the accelerometer values that the sensor gave.

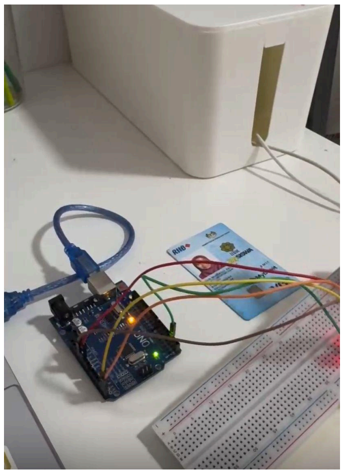
The Python software tested each data input against the requirements for thresholds in real-time while the data was being transmitted from the Arduino to the PC during the data collection phase. The gesture was divided into categories and recorded appropriately if the accelerometer data satisfied either Gesture 1 or Gesture 2 requirements. To ensure that only new movements were recorded, the software kept the last gesture it had identified, which improved output clarity and decreased redundancy. For fundamental movement patterns, our threshold-based approach made gesture detection comparatively easy and successful.

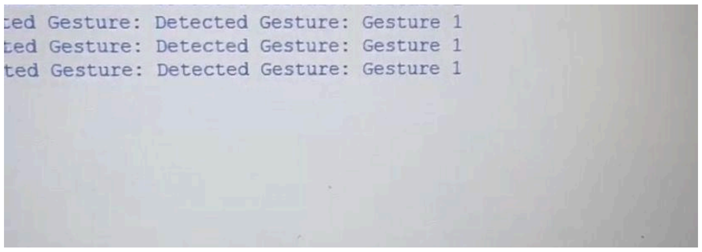
RESULT





```
Python 3.13.0 (tags/v3.13.0:60403a5, Oct  7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:\Users\ASUS\Desktop\System Integration\Experiment_4A\exp_4A2.py ==
Received data: MPU6050 connection successful
Received data: Detected Gesture: Gesture 1
Action for Gesture 1
Received data: Detected Gesture: Detected Gesture: Gesture 2
Received data: Detected Gesture: Detected Gesture: Gesture 2
Received data: Detected Gesture: Detected Gesture: Gesture 2
Received data: Detected Gesture: Detected Gesture: Gesture 2
Received data: Detected Gesture: Detected Gesture: Gesture 2
Received data: Detected Gesture: Detected Gesture: Gesture 2
Received data: Detected Gesture: Detected Gesture: Gesture 2
Received data: Detected Gesture: Detected Gesture: Gesture 2
```





```
ted Gesture: Detected Gesture: Gesture 1
ted Gesture: Detected Gesture: Gesture 1
ted Gesture: Detected Gesture: Gesture 1
```

DISCUSSION

1. Interpretation of result

   <u>Arduino Code</u>
   - Reads accelerometer and gyroscope data from MPU6050 sensor and sends to python code to print the data.
   - Gesture 1 for positive axis and Gesture 2 for negative axis

   <u>Python Script</u>
   - Receive and print the data according to the movement of the sensor.

When the sensor were tilt, the data will be printed (Gesture 1 or Gesture 2) according to the movement of sensor

2. Implication of the result

   - This can be used as real time monitoring as we can be alert as soon as the sensor detects any movement. A quick action can be taken if there is something going wrong that mechanical or robot can not handle. As sensors have a lot of use in industry, it makes work monitoring a lot easier since people only need to monitor the system.

   - Collecting data is important as we can gain a lot from data itself. For example, the rising water can be recorded and used to predict floods. Data also is essential for experiment and research.

3. Discrepancies between expected and observed outcomes.

   Noise or interference in communication

   - Expected Outcome: The serial communication line should be free from electrical noise or interference.
   - Observed discrepancy: if the serial line is subject to electromagnetic interference or poor connection quality, the data may be corrupted, leading to inconsistent or incorrect values being received.
   - Solution: Use proper shielding for the communication cables, ensure that the physical connections are secure, and minimize electromagnetic interference.

4. Sources of error or limitations of the experiment

   Sensor Resolution and Range

- Error/Limitations: The MPU6050 provides 16-bit data for accelerometer and gyroscope readings, but this data is not always directly representative of real-world units (e.g., degrees, meters per second squared).

  Error: The sensor has limited resolution and dynamic range. This means that very small movements or accelerations may not be detectable or measurable.

  Limitations: The sensor's range might not be sufficient for high precision measurements. For example, the accelerometer can measure up to ±16g, which may be limiting in applications requiring higher precision or broader ranges.

- Source: The inherent hardware limits of the MPU6050 sensor's sensors (accelerometer and gyroscope).

RECOMMENDATION

1) Code Optimization and Modularization
    a) Break down the Arduino code into more modular functions for initialization, data reading, and data transmission. This will make the code more readable and easier to debug.
2) Implement Data Filtering Techniques
    a) Introduce a filtering technique, such as a low-pass filter, to reduce noise in the sensor readings. This will lead to more stable and accurate data.
3) Add real-time Data Visualization
    a) Integrate real-time data visualization in Python to allow students to see the sensor output as the experiment is running. Using libraries like Matplotlib or Plotly in Python can help students better understand the data being generated.
4) Provide a detail connection guide
    a) Include a visual wiring diagram in the lab instructions to help students connect the Arduino and MPU6050 sensor accurately. A schematic will reduce common wiring errors, especially for beginners.
5) Include calibration steps
    a) Add calibration steps to account for sensor biases, which will help in achieving more accurate results. Students should perform these calibration steps at the start of the experiment.
6) Alternative Power Options
    a) Suggest the use of external batteries or alternative power sources for the Arduino if the experiment needs to be conducted away from a computer. This can allow for more mobility and different experimental setups.
7) Add use cases and applications

    a) Include a brief discussion on practical applications of IMUs. For instance IMUs apply in drones, smartphones, or wearable devices. This way the students will more understand the real world relevance of their work with the MPU6050.

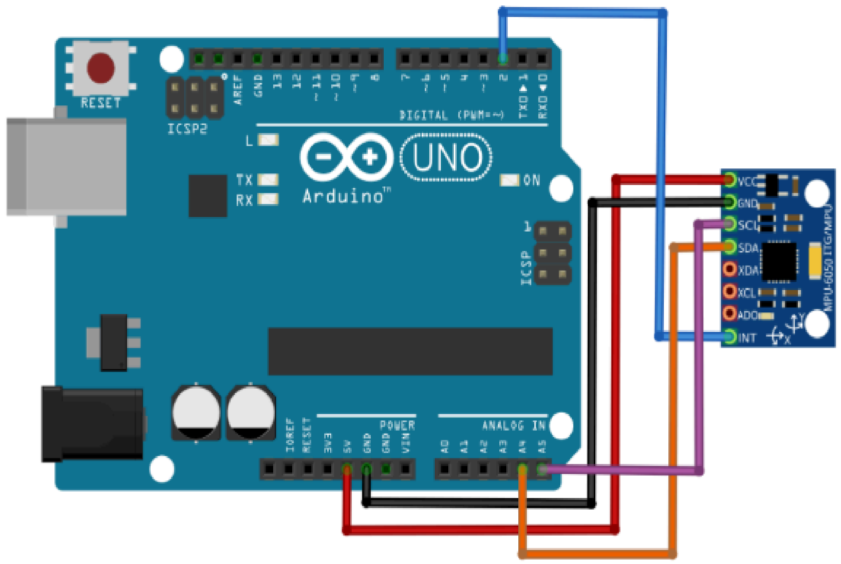
8) Incorporate Error Analysis and Debugging:
    a) Provide a troubleshooting guide for common issues such as sensor drift, connectivity problems, and code compilation errors. A checklist or FAQ can help students troubleshoot on their own.

Insights and Lessons Learned for Future Students

1) Understand the basics of IMU sensors
    a) Familiarize yourself with how accelerometers and gyroscopes work, as this will give you a better grasp of the MPU6050's capabilities and limitations.
2) Importance of calibration
    a) Calibration is key to achieving accurate readings. Spend time on this step and learn about the impacts of drift and bias on sensor data.
3) Debugging is part of the process
    a) Expect to troubleshoot connections, code, and sensor responses. Patience and systematic debugging are crucial skills that will be valuable in any engineering project.
4) Experiment with sensor placement
    a) Placement and orientation of the sensor affect data quality, especially in dynamic setups. Experimenting with sensor positioning can provide better insights into data accuracy.
5) Explore Python Visualization
    a) Python’s visualization tools are extremely useful. Learning how to plot real-time data will be valuable for interpreting sensor readings and understanding how changes in orientation and motion are reflected in data.

APPENDICES

1) Circuit Diagram



Circuit diagram

2) Coding Snippets

a) From Arduino:

```
MPU6050 mpu;

const int threshold = 1000; // Adjust this threshold as needed
int previousGesture = -1;

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

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

```
else {
Serial.println("MPU6050 connection successful");
}
```

Based on the coding that has shown above, we see that we need to import the MPU 6050 library first then we declare the MPU variable first and add the setup before we run it. Then we use the if else statement, If the MPU did not sense any movement then we know that from the output, it will show “MPU6050 connection failed” at the serial monitor. Otherwise, “MPU6050 connection successful” will be shown. Then we know that the connection of the circuit works well. After we check the connection with the Arduino software, we will use Python software to check the gesture.

3) Troubleshooting Tips:

a) Miswiring between Arduino and MPU6050
b) Common error in code
c) Tips for checking and correcting orientation issues

STUDENT 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 that no further improvement on the reports is needed from any of the individual 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 : Muhammad Zamir Fikri Bin Mohd Zamri	**Read**	/
Matric No. : 2212515	**Understand**	/
[illegible] **Signatures :**	**Agree**	/

Name : Muhd Akmal Hakim Bin Saifuddin	**Read**	/
Matric No. : 2216093	**Understand**	/
Signatures : akmal	**Agree**	/

Name : Nur Shadatul Balqish Binti Sahrunizam	**Read**	/
Matric No. : 2212064	**Understand**	/
Signatures : shadatul	**Agree**	/

Name :NORHEZRY HAKIMIE BIN NOOR FAHMY	**Read**	/
Matric No. :2110061	**Understand**	/
Signatures :hezry	**Agree**	/

Name : Nur Amira Nazira Binti Mohd Nasir	**Read**	/
Matric No. :2110026	**Understand**	/
Signatures :amira	**Agree**	/