ДОДАТОК А

ГРАФІЧНИЙ МАТЕРІАЛ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Кваліфікаційна робот
Перший (бакалаврський) рівень

Тіньове відтворення рухів кінцівок людини

Автор:
Сергій Слуту
Студ. Гр. КІУКІ-21-4

Керівник:
Ігор Михайліченко
ас. каф. ЕОМ

План презентації

1. Вступ
2. Апаратна архітектура системи та вибір компонентів
3. Програмна архітектура системи та вибір компонентів
4. Показники точності моделі системи.
5. Результат роботи системи.
6. Відео роботи системи
7. Висновки

1. Вступ

- Проблема:

Відсутність доступних рішень для реабілітації, жестової мови та інтеграції у віртуальні системи.

- Чому це важливо:

Розуміння рухів кінцівок сприяє розвитку технологій для людей з інвалідністю, а також для нових способів взаємодії з технікою.

- Мета:

Створення системи, яка точно зчитує та репрезентує рухи кінцівок.

3

2.1 Вибір компонентів. Arduino Uno.

- Що це:

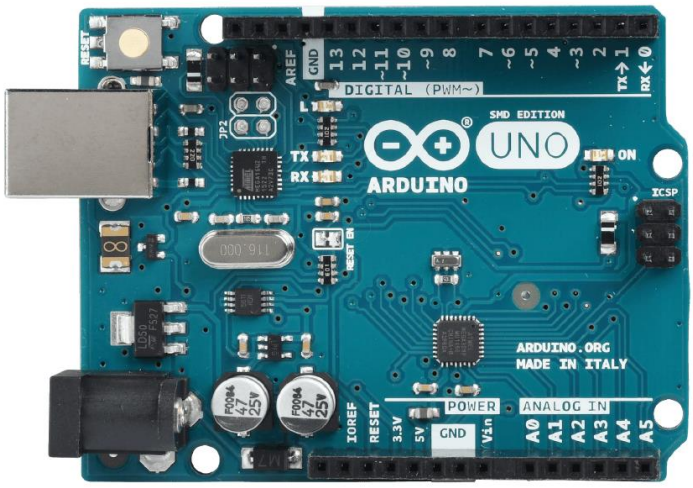
Мікроконтролер для управління електронними компонентами.

- Для чого:

Обробка сигналів із сенсорів та управління системою в реальному часі.

- Чому саме це:

Простота програмування, підтримка великої кількості сенсорів, наявність достатньої кількості входів та виходів.



4

2.2 Вибір компонентів. SpectraFlex Sensors.

- Що це:

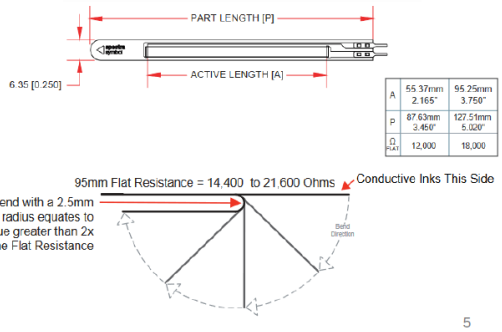
Резистивні датчики, що змінюють опір залежно від згину.

- Для чого:

Вимірювання положення пальців через рівень деформації.

- Чому саме це:

Висока чутливість, низьке енергоспоживання, сумісність з Arduino.



5

2.3 Вибір компонентів. MPU6050.

- Що це:

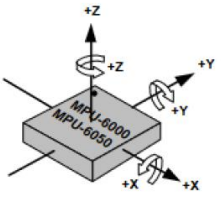
Інтегрований акселерометр і гіроскоп для вимірювання кутових швидкостей і прискорення.

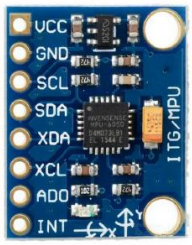
- Для чого:

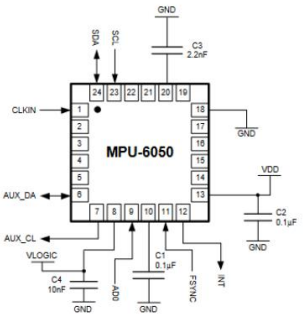
Аналіз положення кисті у просторі.

- Чому саме це:

Компактність, тривісний аналіз руху, підтримка I2C-з'єднання



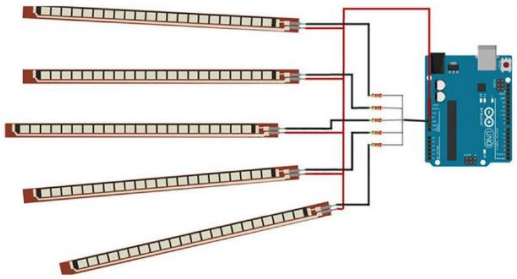




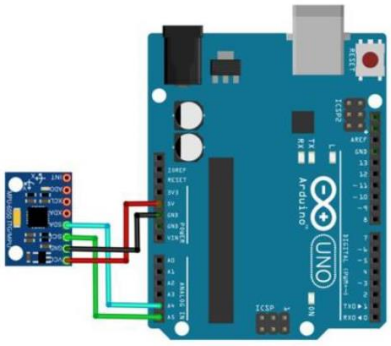
6

3. Апаратна архітектура системи

Підключення Spectra до Arduino.



Підключення MPU6050 до Arduino.



7

3.1 Загальна електрична схема підключення компонентів.

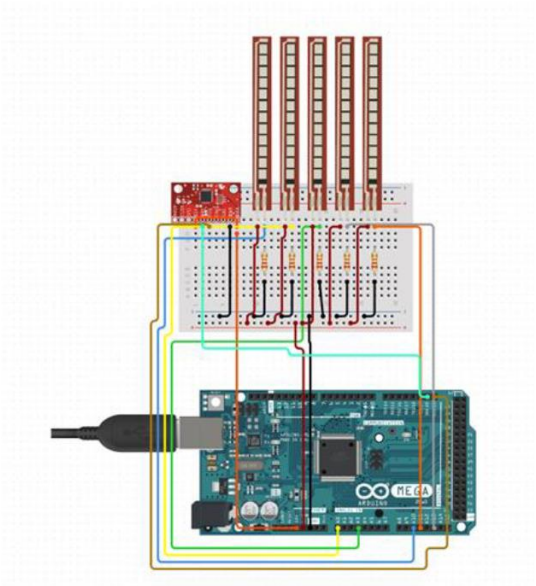
Підключення всіх сенсорів до Arduino Uno.

- Датчики згину

Аналогові входи для вимірювання напруги.

- MPU6050

Інтерфейс I2C для передачі цифрових даних.

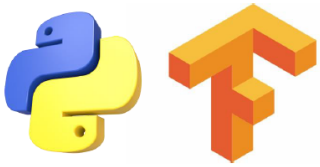


8

4. Програмна архітектура системи. Вибір компонентів.

Python

- Обробка та попередня фільтрація даних із сенсорів
- Злиття та нормалізація сигналів у реальному часі
- Підготовка даних для машинного навчання
- Візуалізація результатів (графіки, діаграми)
- Взаємодія з мікроконтролером через COM-порт



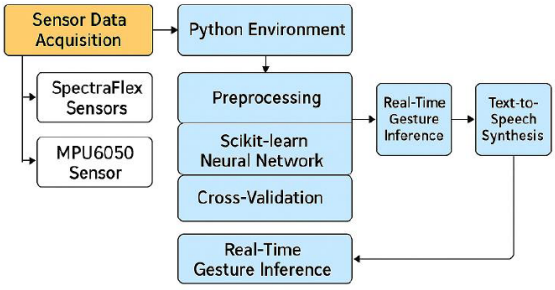
Tensorflow

•Побудова та навчання нейронної мережі
•Класифікація жестів на основі сенсорних даних
•Оптимізація архітектури моделі
•Підтримка інференсу в реальному часі
•Робота з GPU/CPU для пришвидшення обчислень

9

4.1 Загальна флоу-схема роботи системи.

- Дані з гнучких сенсорів SpectraFlex та інерційного сенсора MPU6050 надходять до Python-середовища.
- Там вони проходять попередню обробку, потім — класифікуються нейронною мережею на базі Tensorflow.
- Модель проходить валідацію та виконує розпізнавання жестів у режимі реального часу, після чого результат перетворюється у мову через синтезатор голосу.



10

4.2 Формат даних і обробка.

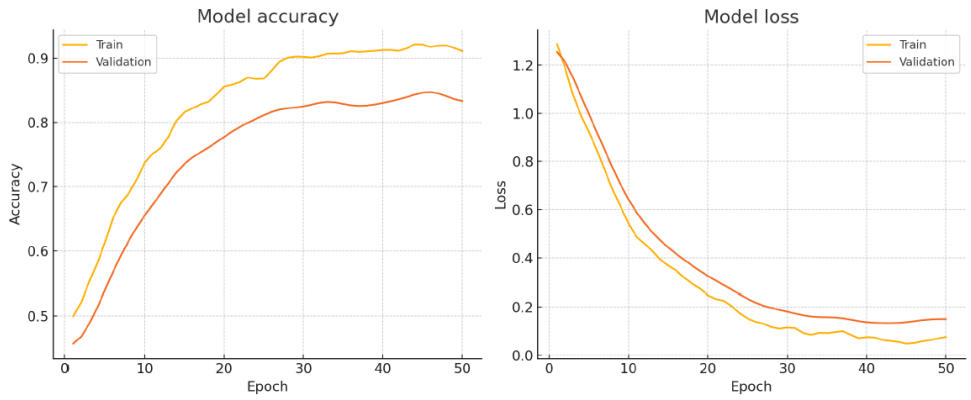
Вхідні дані

AcX	AcY	AcZ	GyX	GyY	GyZ	Thumb	Index	Middle	Ring	Pinky
13520	8688	5940	-6	98	30	132	148	155	161	196
13524	8804	5772	7	49	-18	137	148	155	159	204
13428	8980	5728	-33	42	50	137	148	156	158	201
13296	8860	5848	-241	159	-9	136	148	155	158	200
13288	8832	5752	-148	63	-10	135	148	155	158	199
13620	8884	5812	26	17	-33	136	148	155	158	198
13732	8628	5964	-41	78	-126	135	148	156	158	192
13148	9168	5632	-370	247	9	133	148	157	157	195
13264	8504	6480	-1000	1067	-562	128	148	158	161	187
13676	9520	5584	-118	77	330	128	148	160	161	187
13168	8768	6596	-618	264	-388	124	148	161	163	178
13384	9012	6052	-138	-126	142	125	149	161	162	173

Нормалізовані дані

AcX	AcY	AcZ	GyX	GyY	GyZ	Thumb	Index	Middle	Ring	Pinky
0.901644	0.461077	0.488198	0.57506	0.580956	0.34818	0.227273	0.122222	0.294643	0.648148	0.849057
0.9019	0.467233	0.478383	0.575771	0.574982	0.341561	0.30303	0.122222	0.294643	0.611111	1
0.895737	0.476572	0.475812	0.573584	0.574128	0.350938	0.30303	0.122222	0.303571	0.592593	0.943396
0.887262	0.470204	0.482823	0.562213	0.588393	0.342802	0.287879	0.122222	0.294643	0.592593	0.924528
0.886749	0.468718	0.477214	0.567297	0.576689	0.342664	0.272727	0.122222	0.294643	0.592593	0.90566
0.908064	0.471478	0.48072	0.57681	0.57108	0.339493	0.287879	0.122222	0.294643	0.592593	0.886792
0.915254	0.457893	0.4896	0.573147	0.578517	0.326669	0.272727	0.122222	0.303571	0.592593	0.773585
0.877761	0.486548	0.470203	0.555161	0.599122	0.345284	0.242424	0.122222	0.3125	0.574074	0.830189
0.885208	0.451313	0.519748	0.520719	0.699098	0.266547	0.166667	0.122222	0.321429	0.648148	0.679245
0.911659	0.505227	0.467399	0.568937	0.578396	0.389548	0.166667	0.122222	0.339286	0.648148	0.679245
0.879045	0.465322	0.526525	0.541603	0.601195	0.290541	0.106061	0.122222	0.348214	0.685185	0.509434
0.892912	0.47827	0.494742	0.567844	0.553645	0.363624	0.121212	0.133333	0.348214	0.666667	0.415094

11



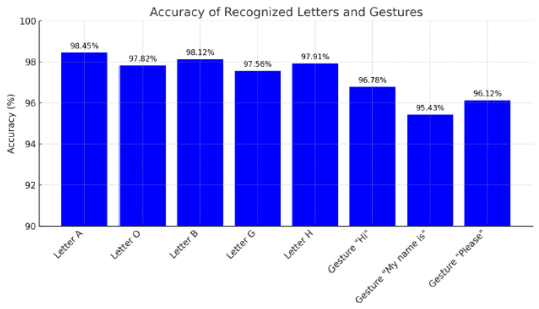
4.3 Показники точності моделі системи.

- Точність моделі (Accuracy) — досягає понад 90% на тренувальній вибірці.
- Втрати моделі (Loss) — поступове зменшення до стабільного рівня після ~30 епох.

12

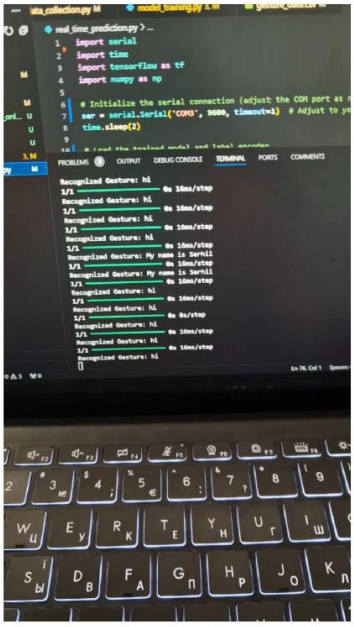
4.4 Результат роботи системи.

- Точність розпізнавання жестів — до 98,4% для окремих літер та понад 95% для складних фраз.
- Модель стабільно розпізнає як окремі символи, так і цілі жести (напр. *Hi*, *My name is*, *Please*).
- Висока узгодженість результатів свідчить про надійність системи в реальному часі.



13

5. Демонстрація роботи системи



14

Висновки

Результати:

•Розроблено функціональну систему для розпізнавання жестів у реальному часі

•Досягнуто високої точності зчитування сигналів (92%+ для сенсорів)

•Нейронна мережа на TensorFlow забезпечує точність класифікації ~73,5%

•Стабільна робота системи підтверджена тривалим тестуванням (95%+)

Майбутні вдосконалення:

•Інтеграція портативного живлення для автономної роботи

•Додавання бездротового з'єднання (наприклад, Bluetooth)

•Оптимізація розпізнавання складних і швидких жестів

•Покращення захисту компонентів для роботи в польових умовах

ДОДАТОК Б
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Б.1 Отримання даних з сенсорів

Б.1.1 Arduino-скетч для зчитування даних

#include <Wire.h>

```
#include <SoftwareWire.h>

const int MPU = 0x68;
int16_t AcX, AcY, AcZ, GyX, GyY, GyZ;
float Tmp;
const int flexPins[] = {A0, A1, A2, A3, A5};
SoftwareWire myWire(2, 3);

void setup() {
  myWire.begin();
  Serial.begin(9600);
  myWire.beginTransmission(MPU);
  myWire.write(0x6B);
  myWire.write(0);
  myWire.endTransmission(true);
  delay(1000);
}

void readMPU6050() {
  myWire.beginTransmission(MPU);
  myWire.write(0x3B);
  myWire.endTransmission(false);
  myWire.requestFrom(MPU, 14, true);

  AcX = myWire.read() << 8 | myWire.read();
  AcY = myWire.read() << 8 | myWire.read();
  AcZ = myWire.read() << 8 | myWire.read();
  Tmp = (myWire.read() << 8 | myWire.read()) / 340.00 + 36.53;
  GyX = myWire.read() << 8 | myWire.read();
  GyY = myWire.read() << 8 | myWire.read();
  GyZ = myWire.read() << 8 | myWire.read();
}

void loop() {
  readMPU6050();

  Serial.print("MPU6050: ");
  Serial.print("AcX = "); Serial.print(AcX);
  Serial.print(" | AcY = "); Serial.print(AcY);
```

```
  Serial.print(" | AcZ = "); Serial.print(AcZ);
  Serial.print(" | GyX = "); Serial.print(GyX);
  Serial.print(" | GyY = "); Serial.print(GyY);
  Serial.print(" | GyZ = "); Serial.print(GyZ);

  const char* sensorLabels[] = {"Thumb", "Index", "Middle",
"Ring", "Pinky"};

  for (int i = 0; i < 5; i++) {
    int sensorValue = analogRead(flexPins[i]);
    Serial.print(" | ");
    Serial.print(sensorLabels[i]);
    Serial.print(" = ");
    Serial.print(sensorValue);
  }

  Serial.println();
  delay(10);
     }
```

Б.1.2 Python-скрипт для збору та збереження даних

```
import serial
import time
import curses
import csv

# Open the serial port with a timeout to prevent blocking
ser = serial.Serial('COM3', 9600, timeout=1)
time.sleep(2)

# Set the target number of samples per gesture
TARGET_SAMPLES = 1000

def read_serial_data():
    try:
        # Read a full line from the serial buffer
        line = ser.readline().decode('utf-8').strip()

        if not line.startswith("MPU6050:"):
            print("Invalid data format, skipping...")
            return None

        # Remove the "MPU6050:" prefix
        line = line.replace("MPU6050:", "").strip()

        # Split the data into key-value pairs
        parts = line.split(' | ')
        data = []

        for part in parts:
            if '=' in part:
```

```
                key, value = part.split('=')
                key = key.strip()
                value = int(float(value.strip()))  # Convert to
float first, then to int
                data.append(value)

        # Ensure we have all 11 data points (6 MPU + 5 Flex)
        if len(data) == 11:
            return data
        else:
            print(f"Incomplete data received, got {len(data)}
values, expected 11.")
            return None

    except Exception as e:
        print(f"Error parsing data: {e}")
        return None

def save_data(data, label, filename='gesture_data.csv'):
    with open(filename, 'a', newline='') as file:
        writer = csv.writer(file)
        for entry in data:
            writer.writerow(entry + [label])

def data_collection(stdscr):
    curses.curs_set(0)
    stdscr.nodelay(1)
    gesture_buffer = []
    label = ""
    sample_count = 0
    recording = False

    try:
        while True:
            stdscr.clear()
            stdscr.addstr(0, 0, "Press 's' to start recording,
'q' to quit.")
            stdscr.addstr(1, 0, f"Recording: {'ON' if recording
else 'OFF'}")
            stdscr.addstr(2, 0, f"Label: {label}")
            stdscr.addstr(3, 0, f"Samples collected:
{sample_count}/{TARGET_SAMPLES}")
            stdscr.addstr(4, 0, "Data:")

            data = read_serial_data()
            if data:
                stdscr.addstr(5, 0, ', '.join(map(str, data)))
                if recording:
                    gesture_buffer.append(data)
                    sample_count += 1

                if sample_count >= TARGET_SAMPLES:
                    save_data(gesture_buffer, label)
```

```
                    stdscr.addstr(6, 0, f"Collected
{TARGET_SAMPLES} samples for gesture '{label}'.")
                    stdscr.addstr(7, 0, "Press 's' to start a
new gesture or 'q' to quit.")
                    recording = False
                    gesture_buffer = []
                    sample_count = 0

            else:
                stdscr.addstr(5, 0, "Invalid data received")

            key = stdscr.getch()
            if key == ord('s') and not recording:
                recording = True
                stdscr.addstr(6, 0, "Enter label for gesture: ")
                stdscr.refresh()
                label = stdscr.getstr(7, 0).decode('utf-
8').strip()
                stdscr.clear()

            elif key == ord('q'):
                if not recording:
                    break
                else:
                    stdscr.addstr(7, 0, "Cannot quit while
recording. Complete the gesture first.")

            stdscr.refresh()
            time.sleep(0.1)

    except KeyboardInterrupt:
        stdscr.addstr(7, 0, "Exiting...")

    finally:
        ser.close()
        curses.endwin()

if __name__ == '__main__':
    try:
        curses.wrapper(data_collection)
    except Exception as e:
        print(f"An error occurred: {e}")
```

Б.1.3 Виправлення pyfirmata для сумісності

```
import os
import sys
import fileinput

# Path to the pyfirmata.py file
pyfirmata_path = os.path.join(sys.prefix, 'Lib', 'site-
packages', 'pyfirmata', 'pyfirmata.py')
```

```
# Replace inspect.getargspec with inspect.getfullargspec
for line in fileinput.input(pyfirmata_path, inplace=True):
    print(line.replace('inspect.getargspec',
'inspect.getfullargspec'), end='')
```

Б.2 Навчання моделі розпізнавання жестів

Б.2.1 Побудова та тренування моделі на TensorFlow

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical

def load_data(filename='gesture_data_normalized.csv'):
    data = pd.read_csv(filename)
    data = data.drop_duplicates()


    # Use only the relevant data
    finger_data =
data[['AcX','AcY','AcZ','GyX','GyY','GyZ','Thumb', 'Index',
'Middle', 'Ring', 'Pinky']].values

    # Scale up finger data to increase its importance (if
desired)
    finger_data = finger_data * 1.5

    X = finger_data
    y = data['Label'].values

    le = LabelEncoder()
    y = le.fit_transform(y)
    y = to_categorical(y)

    return X, y, le

def build_model(input_shape, output_classes):
    model = Sequential([
        Dense(32, input_shape=input_shape, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
        Dropout(0.7),
        Dense(16, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
        Dropout(0.7),
        Dense(output_classes, activation='softmax')
    ])
```

```
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    return model

def train_model(X_train, y_train, X_val, y_val):
    model = build_model((X_train.shape[1],), y_train.shape[1])

    early_stopping =
tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
    lr_scheduler =
tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
factor=0.5, patience=3, min_lr=1e-6)

    history = model.fit(X_train, y_train, epochs=50,
batch_size=32, validation_data=(X_val, y_val),
                        verbose=1, callbacks=[early_stopping,
lr_scheduler])

    return model, history

def cross_validate_model(X, y, n_splits=5):
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
    fold_accuracies = []

    for train_index, val_index in kf.split(X):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        model, history = train_model(X_train, y_train, X_val,
y_val)

        fold_accuracy = history.history['val_accuracy'][-1]
        fold_accuracies.append(fold_accuracy)
        print(f"Fold Validation Accuracy: {fold_accuracy}")

    mean_accuracy = np.mean(fold_accuracies)
    std_accuracy = np.std(fold_accuracies)
    print(f"Cross-Validation Mean Accuracy: {mean_accuracy}")
    print(f"Cross-Validation Std Accuracy: {std_accuracy}")

if __name__ == '__main__':
    X, y, label_encoder = load_data()

    # Splitting the data into training and testing sets
    X_train, X_val, y_train, y_val = train_test_split(X, y,
test_size=0.2, random_state=42, shuffle=True)

    # Train the model on the initial split
    model, history = train_model(X_train, y_train, X_val, y_val)

    # Perform k-fold cross-validation
```

```
    cross_validate_model(X, y)

    # Save the label encoder
    with open('label_encoder.npy', 'wb') as f:
        np.save(f, label_encoder.classes_)

    # Print out the history for inspection
    print("Training complete. Final metrics:")
    print("Accuracy:", history.history['accuracy'][-1])
    print("Validation Accuracy:",
history.history['val_accuracy'][-1])
    print("Loss:", history.history['loss'][-1])
    print("Validation Loss:", history.history['val_loss'][-1])
```

Б.2.2 Розпізнавання жестів у реальному часі

```
import serial
import time
import tensorflow as tf
import numpy as np

# Initialize the serial connection (adjust the COM port as
needed)
ser = serial.Serial('COM3', 9600, timeout=1)  # Adjust to your
serial port
time.sleep(2)

# Load the trained model and label encoder
model = tf.keras.models.load_model('gesture_model_fold_1.h5')
label_encoder = np.load('label_encoder.npy', allow_pickle=True)

# Define a confidence threshold (e.g., 70%)
CONFIDENCE_THRESHOLD = 0.7

def normalize_data(mpu_data, finger_data):
    # Scale up finger data by the same factor as in the training
script
    finger_data = finger_data * 1.5

    # Combine MPU6050 and finger data
    combined_data = np.concatenate([mpu_data, finger_data])

    return combined_data

def read_serial_data():
    try:
        # Read a full line from the serial buffer
        line = ser.readline().decode('utf-8').strip()

        if not line.startswith("MPU6050:"):
            print("Invalid data format, skipping...")
            return None
```

```
        # Remove the "MPU6050:" prefix
        line = line.replace("MPU6050:", "").strip()

        # Split the data into key-value pairs
        parts = line.split(' | ')
        data = []

        for part in parts:
            if '=' in part:
                key, value = part.split('=')
                key = key.strip()
                value = int(float(value.strip()))  # Convert to
float first, then to int
                data.append(value)

        # Ensure we have all 11 data points (6 MPU + 5 Flex)
        if len(data) == 11:
            # Extract MPU6050 data (first 6 values: AcX, AcY,
AcZ, GyX, GyY, GyZ)
            mpu_data = np.array(data[:6])

            # Extract finger data (last 5 values: Thumb, Index,
Middle, Ring, Pinky)
            finger_data = np.array(data[-5:])

            # Normalize data using the same scaling as during
training
            combined_data = normalize_data(mpu_data,
finger_data)

            return combined_data
        else:
            print(f"Incomplete data received, got {len(data)}
values, expected 11.")
            return None

    except Exception as e:
        print(f"Error parsing data: {e}")
        return None

def predict_gesture(data):
    # Predict the gesture using the model
    predictions = model.predict(data.reshape(1, -1))

    # Get the confidence level and the corresponding gesture
    confidence = np.max(predictions)
    gesture = label_encoder[np.argmax(predictions)]

    # Only return the gesture if the confidence is above the
threshold
    if confidence >= CONFIDENCE_THRESHOLD:
        return gesture
    else:
```

```
        return "No gesture detected"  # or you could return None

if __name__ == '__main__':
    try:
        while True:
            # Read the serial data
            data = read_serial_data()
            if data is not None:
                # Predict the gesture
                gesture = predict_gesture(data)

                # Print the recognized gesture if one is
detected
                if gesture != "No gesture detected":
                    print(f"Recognized Gesture: {gesture}")
                else:
                    print(gesture)

            # Short delay to prevent overwhelming the serial
port
            time.sleep(0.1)

    except KeyboardInterrupt:
        # Close the serial connection when the program is
interrupted
        ser.close()
```