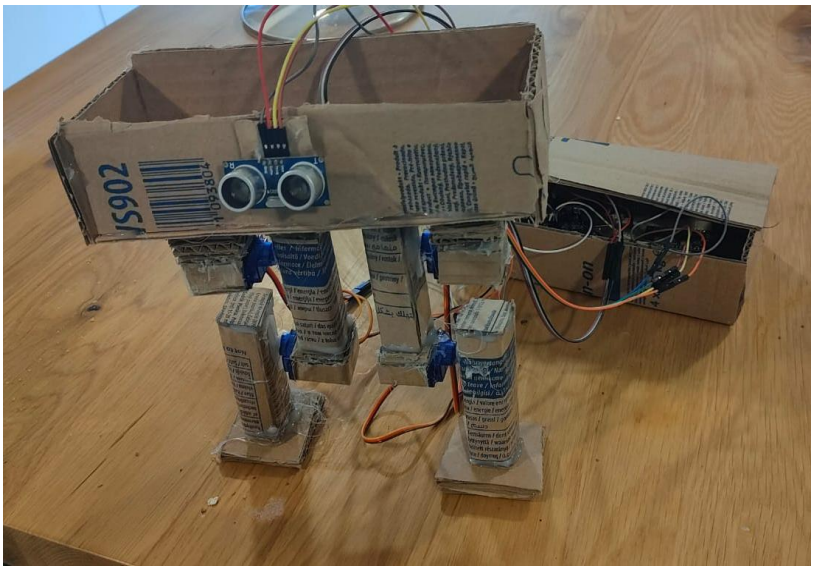
Design and Implementation of an Advanced Microcontroller-based Embedded System



Gait improving Exoskeleton for Stroke Patients.

Date: 06/05/2025 **Module**: EEE416

Lecturers: Dr Sharatchandra Varma Bogaraju, Dr Gabriel Goncalves Machado

Team Members: Sai Adharsh Babu (B00940413), Komal Bairwa (B00925830)

Table of Contents

Abstract

This project presents a wearable robotic exoskeleton for gait rehabilitation in stroke and cerebral palsy patients, covering its design, implementation, methodology, obstacles faced and proposed advancements. This prototype uses an Arduino Uno to process real-time sensor data to control motor output. It focuses on hip and knee movement assistance via the use of servo motors. This device is intended as a cost-effective, accessible rehabilitation tool usable in clinical and home settings.

Introduction

Stroke and cerebral palsy rank among the leading global causes of long-term physical disability. Difficulty with mobility significantly impacts people with these conditions, restricting their independence and negatively affecting their emotional, social, and overall well-being.

Gait rehabilitation is crucial for recovery. Unfortunately, traditional physical therapy usually demands rigorous, repetitive exercises with constant monitoring, a process that might be impractical because of financial, environmental, or resource limitations. For rehabilitation, wearable robotic exoskeletons are proving transformative in this context. Adaptive and consistent motor assistance from this project facilitates the repetitive movement crucial for rehabilitation.

This project focuses on creating an affordable, wearable robotic exoskeleton for stroke patients and individuals with cerebral palsy. Using motorized joints and real-time feedback, it aids hip and knee movement. The prototype's goal is to close the gap between professional clinical care and home rehabilitation. This project improves access to personalized gait training, helping users regain mobility and control over their lives.

Design Specifications

The project should involve I/O interfacing, use of communication interfaces, use of interrupts and data processing:

1. General I/O Interfacing: The user should be able to key in variables needed for control of the embedded system. Sensors or keypads can be used as input devices.

2. Interrupts: The embedded system should have user-activated interrupt or sensor activated interrupt. User-activated interrupt can be done using a switch or button.

3. Communication: The embedded system should use communication protocols to transfer data between two microcontroller boards.

4. Data Processing: The user should be able to store at least 10 different data points either given by the user or by sensors at different points in time. The embedded system should show some statistics based on the data collected.

Potential Designs

Design 1 - Infant Vein Finder using Arduino-based Technology

The first design is a non-invasive, portable device using near-infrared (NIR) light technology and an infrared camera to locate and visualise the hard-to-find veins in infants' arms. The differing absorption of infrared light by veins and surrounding tissue allows them to be seen on OLED displays or smartphone apps. This lowers pian and risk by reducing the number of needle insertions. Healthcare providers can use this system to quickly and safely locate veins for procedures such as blood sampling and IV insertion.

Design 2 - Thought-to-Text System via EEG

The second design uses an EEG headset, Arduino, and machine learning to convert brain signals into text. This project allows those with severe speech or motor impairments to use hands-free communication.

Design 3 - Smart Wearable Glove to Detect Early Parkinson's Disease

The third design is a wearable sensor glove that detects micro hand tremors, which are early indicators of Parkinson's disease. This design aids in early Parkinson's diagnosis to enable timely intervention. The collected data helps track disease progression and make early diagnoses. Adaptable for home use, it reduces the need for frequent clinic visits.

Design 4 – Intelligent Rehabilitation Gloves:

The fourth design is a glove designed for stroke-recovering patients, traumatic brain injury, or hand surgery, and helps the user restore motor control through exercises and real-time feedback. Its lightweight, user-friendly design enhances daily usability by providing quantitative progress data.

Final Design

Exoskeleton for gait improvement assisting stroke & cerebral palsy patients in mobility recovery.

This is a wearable robotic exoskeleton that assists in the recovery of walking for stroke and cerebral palsy patients. The device aids patients in gait training by assisting their hips and knees. Sensor input is interpreted by an Arduino Uno and adjusts motor output in real-time. This project aims to improve recovery after neurological injury, and can be used in clinical or home-based environments for continuous rehabilitation.

Methodology

Application

This project is a wearable robotic exoskeleton system built using Arduino, NodeMCU (ESP8266), servo motors, sensors, and cloud-based control and prediction systems. It allows for IoT-based control through voice commands using Google Assistant, IFTTT, Webhooks, and a Flask server with an ML model. It features real-time movement control, obstacle detection, and intelligent movement pattern management.

System Components Breakdown

1. Arduino Uno

It Controls the operation of the servo motors and receives input from the various sensors (Ultrasonic and MPU6050).

Connections:

- Servo Motors: Connected to digital PWM pins: D3, D5, D6, D10
- Ultrasonic Sensor: Trig → D8, Echo → D9
- MPU6050 Gyroscope: SDA → A4, SCL → A5

2. NodeMCU (ESP8266)

It is a Wi-Fi module and IoT module responsible for sending POST requests to the the Flask server as well as receiving commands to control movement

Pin Configuration: None, it simply connects via USB serial to receive anticipated movement commands from the machine learning model.

3. Flask Server (Running on PC)

Flask is a micro web framework for Python which allows for the hosting of the trained Machine Learning model on a local server (on an external laptop). The Flask server creates an API endpoint (/predict) which can be reached through an HTTP request which sends data from the ultrasonic sensor and MPU6050 to predict Movement and send that back.

Working:

- Uses an HTTP POST request to send and receive data
- Loads .pkl ML model to predict class of movement
- Sends back predicted movement command

4. Google Assistant + IFTTT + Webhooks

It is a voice-controlled command system which converts voice commands to Webhook requests to be processed by the flask server.

Voice Commands Used:

- **Start:** The exoskeleton moves from a sat resting position to a neutral standing position. It sets all four servos (two per leg) to 90 degrees, resulting in a straight and upright leg position.
- **Stop:** The exoskeleton rotates all servos to 0 degrees and folds the leg into a resting pose.
- **Start Walking:** This command begins a walking sequence by alternating servo angles to simulate human movement. This is done in a loop with alternating leg postures to simulate step-forward movements.
- **Left and Right Turn:** The servos on both legs asymmetrically adjust to emulate a turning sequence. To turn right, the right leg moves forward and the left leg moves back which causes the torso to rotate slightly to the right. A set number of repetitions of this will turn the exoskeleton to the desired direction.
- **U-Turn:** The exoskeleton performs multiple right turns to complete a full 180-degree revolution.

- **Start/Stop Dance:** This command triggers A fun dance that involves lifting and lowering one leg twice, followed by the other leg in a loop. The "Stop Dance" command ends the cycle and returns the servos to their neutral position.

Command Logic and Motor Movement

Each command is associated with a movement pattern driven by controlling the servo motors using servoX.write(angle) commands, where the angle sets the position of the servo.

1. **Start Prototype**
    - The device moves from a sitting position to a neutral standing pose.
    - Motor Action: All four servo motors are set to a 90° angle and the legs are in an upright position.

2. **Start Walking**
    - This command initiates a walking cycle from a standing position.
    - The steps are alternative and thee legs rotate between the foeward phase
    - Motor Action: If the right leg moves forward (servo1,3), then the left leg (servo 2, 4) will remain in the same position and then this alternates, simulating walking. Each step includes a forward phase and a pause (reset to 0°).

3. **Stop Walking**
    - This command resets all the servos to a sitting/resting state.
    - Motor Action: All servos set to 0°, lowering the legs.

4. **Turn left / Turn right**
    - This command makes the system rotate in in place to turn to the desired direction.
    - Motor Action:
        - turn right: right leg forward, left leg back → servo1/3 = 90°, servo2/4 = 0°
        - turn left: left leg forward, right leg back → servo2/4 = 90°, servo1/3 = 0°

5. **U-Turn**
    - This command executed a 180 degree rotation in used repeated turn right commands to achieve that
    - Motor Action: Repeats turn right sequence 10 times.

6. **Start Dance**
    - This command makes the exoskeleton do alternating leg movements that resemble dancing.
    - Motor Action:
        - Right leg up/down twice → servo1/3 toggle 0° and 90°
        - Left leg up/down twice → servo2/4 toggle 0° and 90°
        - Repeats until stop dance is triggered.

7. **Stop Dance**
    - Description: Terminates ongoing dance routine.
    - Motor Action: Halts any ongoing motor commands.

Sensor Feedback and Adaptive Control:

The MPU6050 provides six-axis motion data (acceleration and gyroscope), to the Arduino through the I2C. It sends angular velocity and linear acceleration values for each of the 3d axes. It can also be used for future prospects of the device like monitor leg posture, detect fall risk, or trigger recovery mechanisms.

Example: If the MPU6050 detects a sudden forward tilt, the servos can react to shift weight back or lock joints for stability. This will make the exoskeleton smart and adaptable to unexpected changes in user movement or the structure of the terrain.

Obstacle Detection System:

The exoskeleton includes an obstacle avoidance mechanism using an ultrasonic sensor (HC-SR04) and an MPU6050 gyroscope. The sensor detects obstacles in the path and automatically triggers a manoeuvring routine when an object is closer than 20 cm.

Sensor Used:

- Ultrasonic Sensor (HC-SR04)
- Pins: TrigPin = 8, EchoPin = 9

How it works:

- The Ultrasonic sensor sends a sound pulse from the trigPin.
- It then receives the sound that bounced back through the echopin and measures the duration.
- Distance formula [distance = duration * 0.034 / 2]

Avoidance Logic:

- The ultrasonic sensor sends out sound pulses and calculates the disctance between the objects using the bounced back waves.
- If an object is detected within 20 cm, the exoskeleton stops and turns right by 45 degrees to start the obstacle clearance sequence. The exoskeleton will start walking forward alongside the object while measuring its length. Once it reaches the end of the obstacle, it moves for a further distance of 10 cm, which adds a clearance distance for the object to turn. After the device clears the obstacle, it turns back to the original direction and continues.

Movement Pattern:

- Each forward movement assumed ~5cm
- Obstacle tracking loop estimates bypass distance

Webhooks, IFTTT & Google Assistant

What is IFTTT?

“If This Then That” is a cloud platform that is used for connecting apps and devices using conditional statements. In the exoskeleton, it is used to link the user voice commands with the action movement sequences of the device.

What are Webhooks?

Webhooks are HTTP URLs that accept data and trigger responses in real time. Each command uses a unique Webhook.

Workflow:

1. The User Command is recorded by Google Assistant. Example: Google Assistant hears: "Start Walking"
2. This action triggers the IFTTT Applet, and as a result Google Assistant triggers the Webhook.
3. The Webhook will send POST Request and it will be received by the Flask Server as a Flask endpoint /voice_command.
4. Flask Server reads and interprets the command. The interpreted command in this case will be Parses JSON { "command": "start walking" }.
5. Now the Server sends this to the Serial Port and the serial port transfers this command to the Arduino.
6. The Arduino receives the command and executes the appropriate motor logic.

Example IFTTT Setup:

- **Trigger Phrase**: "Start Walking"
- **Webhook URL**: http://192.168.1.100:5000/voice_command
- **Method**: POST
- **Body**:

```
{
  "command": "start walking"
}
```

Flask Server + ML Model

Flask Server

- Lightweight Python web framework
- Two primary routes:
    - /voice_command → Receives IFTTT voice commands
    - /predict → Accepts sensor data from NodeMCU

ML Model

- Trained model in model.pkl using train_model.py
- Takes gyroscope/accelerometer features from MPU6050
- Predicts movement intent (e.g., walk, stop, dance)

Prediction Workflow:

1. **NodeMCU** sends sensor data to /predict
2. **Flask** loads model.pkl and predicts movement
3. **Flask** sends prediction to Arduino over serial
4. **Arduino** updates motor state accordingly

Pin Mapping Summary

Component	Pin (Arduino)	Description
Servo 1	D3	Right Leg Front
Servo 2	D5	Left Leg Front
Servo 3	D6	Right Leg Rear
Servo 4	D10	Left Leg Rear
TrigPin (Ultrasonic)	D8	Ultrasonic pulse trigger
EchoPin (Ultrasonic)	D9	Echo pulse receiver
MPU6050 SDA	A4	I2C data line
MPU6050 SCL	A5	I2C clock line
NodeMCU to PC	USB	Sends/receives JSON movement data

Information Flow: Step-by-Step

1. **Initialization**
    a. Arduino initializes servos and sensors
    b. NodeMCU connects to Wi-Fi
    c. Flask server loads model and starts on port 5000
2. **Voice Command Sequence**
    a. User speaks → Google Assistant activates IFTTT applet

b. Webhook sends command to Flask
c. Flask parses and forwards command to Arduino

3. **Sensor-Driven ML Prediction**
    a. NodeMCU reads real-time MPU6050 data
    b. Sends JSON to Flask /predict
    c. ML model returns movement intent
    d. Command passed to Arduino
4. **Motor Control Execution**
    a. Arduino interprets command
    b. Updates servo angles as per logic
5. **Obstacle Handling**
    a. Ultrasonic sensor detects proximity
    b. Initiates object bypass logic sequence
    c. Resumes previous command upon clearance

Project Code:

This section explains the code of the program. The code is written manually and refined using ChatGPT. The project uses three separate codebases:

1. **Arduino Uno Code:** This code runs on the Arduino Uno and handles the basic movement patterns and loop sequences for the demo model.
2. **ESP8266 Code:** This code is uploaded to the ESP8266 module. It sends HTTP POST requests to the Flask server to request movement commands and receive instructions.
3. **Flask Server Code:** This Python-based code runs locally on a PC using the Flask framework. It hosts a trained machine learning model to predict movement patterns based on incoming sensor data. It also processes HTTP POST requests and receives commands from IFTTT Webhooks, which are then forwarded to the ESP8266, and then to the Arduino Uno for execution.

Arduino Code:

The code will be broken down and analysed in sections begin by firstly analysing the Global section which starts by including the outside libraries required for the operation of the servos as well as connections to the Arduino Uno Board and the MPU 6050.

```
#include <Servo.h>

#include <Wire.h>

#include <MPU6050.h>
```

This part of the code declares the servos. hipLeft and kneeLeft if for Left leg movement and hipRight and kneeRight is for right leg movement.

```
Servo hipRight, hipLeft, kneeRight, kneeLeft;MPU6050 mpu;
```

This Part of the code defines the pins for the ultrasonic sensor. The trigPin sends the pulse and the echoPin receives the pulse.

```
const int trigPin = 8;
const int echoPin = 9;
```

Variables for System Control

```
MPU6050 mpu;
bool systemStarted = false;
bool isDancing = false;
String command = "";
```

The MPU6050 handles both the gyroscope and accelerometer functions, while the systemStarted variable tracks whether the device is active. The isDancing variable indicates if the dancing mode is currently enabled, and the command variable stores incoming serial commands.

Setup Function

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

Initialises serial communication for command input.

```
  hipRight.attach(3);
  hipLeft.attach(5);
  kneeRight.attach(6);
  kneeLeft.attach(10);
```

Connects each servo to its respective pin.

```
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
```

Sets ultrasonic sensor pins.

```
  Wire.begin();
  mpu.initialize();
  standNeutral();
}
```

Initialises the I2C and the MPU6050 and sets the device to neutral standing position.

Loop Function:

```
if (Serial.available()) {
  command = Serial.readStringUntil('\n');
  command.trim();
```

The program checks whether a new serial message has been received, then reads the message and trims any trailing spaces or newline characters.

```
  if (command == "start") {
```

```
    systemStarted = true;
    standNeutral();
```

Activates the device and puts it in a ready pose.

```
  } else if (command == "stop") {
    systemStarted = false;
    sitDown();
```

It deactivates the device and puts it in the resting pose.

```
  } else if (systemStarted) {
    // Only accept commands if system is active
    if (command == "start walking") walkSteps();
    else if (command == "turn left") turnLeft();
    else if (command == "turn right") turnRight();
    else if (command == "u turn") {
      turnRight(); turnRight(); turnRight(); turnRight();
    }
    else if (command == "dance") isDancing = true;
    else if (command == "stop dance") isDancing = false;
  }
}
```

Once the device has started, various movement patterns are enabled. Some of which are start walking, turn right and start dancing.

```
if (systemStarted) {
  if (detectObstacle()) {
```

```
    avoidObstacle();
  }

  if (isDancing) {
    danceRoutine();
  }
}
```

The device checks for nearby obstacles using the ultrasonic sensor when the system is activated. When an obstacle is detected, it calls the avoidObstacle() function to navigate around it. If the isDancing flag is set to true, the robot performs a dance routine instead.

Body Position Functions

```
void standNeutral() {
  hipRight.write(90);
  hipLeft.write(90);
  kneeRight.write(90);
  kneeLeft.write(90);
}
```

It centres all joints at 90°, making the device stand upright in a neutral position.

```
void sitDown() {
  hipRight.write(0);
  hipLeft.write(0);
  kneeRight.write(0);
  kneeLeft.write(0);
}
```

It folds the device's limbs and effectively puts it into a "rest" pose.

Walking Behavior

```
void walkSteps(int steps) {
  for (int i = 0; i < steps; i++) {
    // Right leg forward
    hipRight.write(60);
    kneeRight.write(120);
    delay(400);
    standNeutral();
    delay(300);

    // Left leg forward
    hipLeft.write(60);
    kneeLeft.write(120);
    delay(400);
    standNeutral();
    delay(300);
  }
}
```

The device imitates the movement of a walking gait by alternating between moving its right and left legs. The robot swings one leg forward at the hip to a 60° angle while bending the knee to 120° to take a step forward. After completing the motion, the leg returns to a neutral position. The robot then switches to the other leg and repeats the process. This cycle continues until the next command is issued.

Turning Functions:

```
void turnRight() {
  hipRight.write(60);
  kneeRight.write(120);
  hipLeft.write(120);
  kneeLeft.write(60);
  delay(500);
  standNeutral();
  delay(300);
}
```

To rotate the robot to the right, the right leg is moved slightly forward, and the left leg slightly backwards (and vice versa with the knees), and this offset motion causes the body to rotate right. After this, it returns to the neutral position.

```
void turnLeft() {
  hipLeft.write(60);
  kneeLeft.write(120);
  hipRight.write(120);
  kneeRight.write(60);
  delay(500);
  standNeutral();
  delay(300);
}
```

The left turn applies the same principle as the right turn, but in opposite directions.

Dance Routine:

```
void danceRoutine() {
  hipRight.write(60);
  kneeRight.write(120);
  delay(300);
  standNeutral();
  delay(300);
  hipLeft.write(60);
  kneeLeft.write(120);
  delay(300);
  standNeutral();
  delay(300);
}
```

The device performs a simple two-part dance move by first moving its right leg in a dance step and pausing briefly. It then repeats the same motion with the left leg. This sequence continues for as long as the isDancing condition remains true.

Obstacle Detection with Ultrasonic Sensor

```
bool detectObstacle() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);
  float distance = duration * 0.034 / 2;
```

```
  return distance < 20;
}
```

The device uses its ultrasonic sensor to transmit a sound pulse and then measures how long it takes for the echo to return. This time is then converted into a centimetre-based distance measurement. If the observed object is less than 20 centimetres away, the function returns true.

Smart Obstacle Avoidance

```
void avoidObstacle() {
  Serial.println("Obstacle Detected! Avoiding...");
```

This activates when an obstacle is close.

Obstacle Clearence Strategy:

Turn Right 90°:

```
for (int i = 0; i < 4; i++) {
  turnRight();
}
```

Rotates by doing right turns.

Walk Along the Obstacle:

```
int steps = 0;
while (detectObstacle()) {
  walkSteps(1);
  steps++;
  delay(200);
}
```

Walks alongside the obstacle until it detects no obstacle ahead, and also measures the number of steps taken.

Buffer Distance:

```
walkSteps(6);
```

The device walks another 6 step to ensure that the object can clear the obstacle without any collision.

Turn Back Left to Realign to Original Direction:

```
for (int i = 0; i < 4; i++) {
  turnLeft();
}
```

Resume Original Path:

```
walkSteps(steps + 6);
```

Walks the same total distance on the original path to stay aligned with its route.

ESP 8266 Wifi (Node MCU) Code:

The ESP 8266 code will be broken down and analysed in this section. Let us begin by first analysing the Global libraries.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
```

The ESP8266WiFi.h library enables your ESP8266 microcontroller to connect to a Wi-Fi network. The ESP8266HTTPClient.h library allows the microcontroller to send HTTP requests, such as POST, to web servers. Together, these libraries allow the ESP8266 to function as a network-connected client capable of exchanging data with a backend server, like Flask.

```
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";
```

```
const char* serverUrl = "http://<FLASK_PC_IP>:5000/predict";
```

In this part of the code, we have to replace "YOUR_SSID" and "YOUR_PASSWORD" with the Wi-Fi credentials of the network the ESP8266 will be connected to, and the <FLASK_PC_IP> will be replaced by the IP address of the Pc that is running the Flask server on port 5000.

setup()

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

Starts the serial communication at 9600 baud and print logs to the Serial Monitor.

```
  WiFi.begin(ssid, password);
```

Starts the connection to the specified Wi-Fi network.

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
```

Checks the connection every second and prints connecting to Wifi until a connection is established.

loop()

```
String data = "{\"gyro\": [0.1, 0.2, 0.3], \"ultrasonic\": 15}";
```

This is the JSON data that will be sent to the server. The “gyro” simulates the gyroscope values and the “ultrasonic” measures the distance.

Sending Data to Flak Server:

```
if (WiFi.status() == WL_CONNECTED) {
  HTTPClient http;
  http.begin(serverUrl);
  http.addHeader("Content-Type", "application/json");
```

The program checks if the Wi-Fi connection is still active. It then initialises an HTTPClient object and begins a connection to the Flask server URL. The request header is set to indicate that the body content is formatted as JSON.

```
  int httpResponseCode = http.POST(data);
```

The program sends an HTTP POST request with the JSON payload (data) to the server. The server will now process it and return a prediction.

```
  if (httpResponseCode == 200) {
    String response = http.getString();
    Serial.println(response);  // Send to Arduino via serial
  }
```

If the server responds with HTTP 200 (OK), the ESP8266 reads and prints the response to the Serial Monitor.

```
  http.end();
}
```

Frees up the resources used by the HTTP connection.

```
delay(2000);
```

The program waits for 2 seconds before sending the next request to avoid flooding the server.

Flask Server Code:

Importing Libraries:

```
from flask import Flask, request, jsonify
import pickle
import serial
```

Flask is a lightweight Python web framework used to create web servers. The request object provides access to incoming data, such as JSON sent by the ESP8266. The jsonify function converts Python dictionaries into JSON-formatted responses. The pickle module is used to load pre-trained machine learning models from files, and lastly, the serial module enables communication with an Arduino over a serial / USB connection.

Initializing Flask App and ML Model

```
app = Flask(__name__)
model = pickle.load(open("model.pkl", "rb"))
```

The app = Flask(__name__) is used to start the Flask application. The model is loaded from a model.pkl file.

Setting Up Serial Communication with Arduino

```
ser = serial.Serial('COM3', 9600)
```

A serial connection is established with the Arduino on port COM3 at a baud rate of 9600.

Prediction Endpoint – /predict

```
@app.route('/predict', methods=['POST'])
```

```
def predict():
```

This route handles prediction requests (from ESP8266).

```
    data = request.get_json()
    features = [data['gyro'][0], data['gyro'][1], data['gyro'][2], data['ultrasonic']]
```

Extracts three values for gyro and a distance value for ultrasonic.

```
    prediction = model.predict([features])[0]
```

The features are fed into the machine learning model, which then returns the first prediction.

```
    ser.write((prediction + '\n').encode())
```

The command is sent to the Arduino over the serial connection, with a newline character (\n) appended to ensure that Serial.readStringUntil('\n') on the Arduino correctly detects the end of the message.

```
    return jsonify({'command': prediction})
```

The program sends back a JSON response to the client.

Voice Command Endpoint – /voice_command

```
@app.route('/voice_command', methods=['POST'])
def voice_command():
```

This is an alternative approach where voice-command-style strings, such as "start walking" or "stop walking", can be sent directly to the system. This bypasses the need for sensor-based input or model predictions.

```
    data = request.get_json()
    command = data.get("command", "").lower()
```

This converts the commands to lower case for consistency.

```
    ser.write((command + '\n').encode())
    return jsonify({"status": "success", "received": command})
```

The command is sent to the Arduino, and the program returns a success message.

Running the Server:

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

The program runs the Flask app on all available network interfaces at port 5000.

Circuit Design

This section will explain the construction of the device's circuitry, component attachment and powered operation. A circuit diagram is provided to offer insight into wiring and arrangement complexities.

Servo Motors:

It is powered by an external power supply as they cannot draw too much power from the Arduino.

A physical switch between power supply and servos for an emergency stop.

Connected to Arduino as follows:

Servo 1 (Right Leg Front): D3

Servo 2 (Left Leg Front): D5

Servo 3 (Right Leg Rear): D6

Servo 4 (Left Leg Rear): D10

Ultrasonic Sensor:

It is powered by Arduino.

Trig pin: D8

Echo pin: D9

MPU6050 Gyroscope/Accelerometer:

It is also powered by the Arduino.

Communicates via I2C:

SDA (Data Line): A4

SCL (Clock Line): A5

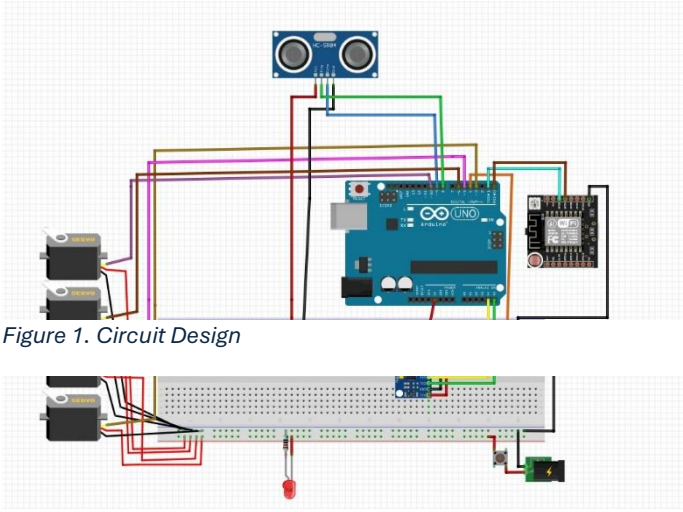


Figure 1. Circuit Design

NodeMCU (ESP8266):

It is powered through a USB plug that is connected to the PC.

Functionality:

Responsible for transmitting and receiving JSON movement data (with the backend, i.e. Flask server).

Power Sources:

Arduino and Nodemcu are powered through USB power cables that are connected to the computer.

Proposed Power Source:

The proposed power source was a battery for a standalone, mobile configuration.

The modal became too heavy from adding the battery and couldn't move. Thus, an external power source was used to prevent it and stabilise the robot for better functionality.

Obstacles

Throughout this project we faced many obstacles, including the following:

Design Modifications:

In the initial stages of development, the circuit board was mounted onto the top of the physical model. This configuration, however, was structurally unstable because the extra weight made the model repeatedly fall. Clearly, the framework couldn't support the circuit board unless reinforced. To resolve this problem the circuit board was detached from the model and placed externally. The model's structural integrity improved significantly after this adjustment, which relieved excessive weight.

To further enhance stability, we added materiasl efficiency strategies. We removed nonessential components and used the minimum amount of building materials. The cardboard body was remade and reassembled with minimal components for improved functionality and form, which resulted in a lighter and more balanced model.

Technical Challenges:

This project involved a substantial amount of coding, which presented its own set of challenges. We encountered numerous errors and issues within the code during the development phase, and many of these were unforeseen and time-consuming to troubleshoot. The limited timeframe also added some extra pressure, as we were concerned that any unresolved errors could hinder the successful execution of the program. Despite these difficulties, through constant debugging and testing, we were ultimately able to complete the code and achieve the desired functionality, and the system was successfully executed.

Future Developments

- Ankle Actuation: The addition of servo motors at the ankle joints enables dorsiflexion and plantarflexion, which allows for a more complete gait cycle. It also enhances and

maintains a steady balance during operation. The device will more closely replicate natural biomechanics by actively supporting these movements, thereby enhancing both better rehabilitation outcomes and user comfort during gait training.

- Advanced CAD Design and Materials: The cardboard body of the current prototype was suitable for early testing also had limited durability and structural performance. Future designs will leverage a precisely engineered 3D CAD model for better fit, customization, and manufacturability. The use of lightweight, high-strength materials, such as carbon fibre composites or advanced thermoplastics, improves wearability, load distribution, and also allows for long-term usability.
- Enhanced Sensors: Sensors like IMUS, EMG sensors, and foot pressure sensors to can be integrated into the system to detect weight distribution and gait phases. These enhancements will allow the exoskeleton to better interpret user intent, which will enable more adaptive and individualised assistance that evolves with the patient's progress.
- Upgraded Control Hardware: The Arduino Uno lacks the processing power and I/O for advanced features, and thus, hardware that can do improved processing is required for better control.Hence, using a more powerful microcontroller like a Raspberry Pi instead of an Arduino Uno will allow for wireless communication, real-time data logging, and more advanced control algorithms.
- Machine Learning Integration: The Implementation of adaptive control through machine learning by analysing sensor data over time so that the system can automatically adjust motor outputs and resistance levels. This approach fosters more efficient and patient-specific recovery trajectories.

Appendix

Ulster University **EEE416 Team Meeting Minutes**		
Design Team: Sai Adharsh Babu, Komal Bairwa		**Date: 04/03/2025**
Brief Note of Discussion: Have a look at the design specifications and create any concept ideas based on the specifications.		
Actions	**Team member(s) responsible**	**Deadline**
Create concept ideas.	Sai, Komal	11/03/2025
	Date of next meeting: 11/03/2025	

Ulster University **EEE416 Team Meeting Minutes**		
Design Team: Sai Adharsh Babu, Komal Bairwa		**Date: 11/03/2025**
Brief Note of Discussion: Read through the concept ideas and finalised one – Gait exoskeleton for stroke patients.		
Actions	**Team member(s) responsible**	**Deadline**
Research the finalised concept.	Sai, Komal	19/03/2025
	Date of next meeting: 20/03/2025	

Ulster University **EEE416 Team Meeting Minutes**	
Design Team: Sai Adharsh Babu, Komal Bairwa	**Date: 20/03/2025**
Brief Note of Discussion: Read through the concept ideas and finalised one – Gait exoskeleton for stroke patients. Request access to soldering lab.	

Order components needed from amazon.		
Actions	**Team member(s) responsible**	**Deadline**
Research the finalised concept, coding, circuit design.	Sai, Komal	24/03/2025
Request access to soldering lab.	Sai	21/03/2025
Order the components needed for concept.	Komal	21/03/2025
	Date of next meeting: 02/04/2025	

Ulster University **EEE416 Team Meeting Minutes**		
Design Team: Sai Adharsh Babu, Komal Bairwa	**Date: 02/04/2025**	
Brief Note of Discussion: Soldered gyroscope and accelerometer components in lab. Started coding the program and put the circuit board together.		
Actions	**Team member(s) responsible**	**Deadline**
Soldered gyroscope and accelerometer components in lab.	Sai, Komal	02/04/2025
Coding	Sai	07/04/2025
Circuit Board	Komal	07/04/2025
	Date of next meeting: 10/04/2025	

Ulster University **EEE416 Team Meeting Minutes**		
Design Team: Sai Adharsh Babu, Komal Bairwa	**Date: 10/04/2025**	
Brief Note of Discussion: Sai coded the program, however errors kept arising. Completion of circuit board. Sarted creating the model out of cardboard.		
Actions	**Team member(s) responsible**	**Deadline**
Coding	Sai	14/04/2025
Cardboard model	Komal	14/04/2025
	Date of next meeting: 14/04/2025	

Ulster University **EEE416 Team Meeting Minutes**		
Design Team: Sai Adharsh Babu, Komal Bairwa		**Date: 14/04/2025**
Brief Note of Discussion: Kept coding the program, however errors kept arising. Created the first model out of cardboard.		
Actions	**Team member(s) responsible**	**Deadline**
Coding	Sai	30/04/2025
Cardboard model	Komal	30/04/2025
	Date of next meeting: 30/04/2025	

Ulster University **EEE416 Team Meeting Minutes**		
Design Team: Sai Adharsh Babu, Komal Bairwa		**Date: 30/04/2025**
Brief Note of Discussion: Tested the complete model, including the coding, circuit board and cardboard model in lab. It worked but kept falling due to the weight of the circuit on top of the model .		
Actions	**Team member(s) responsible**	**Deadline**
Make minor changes to the shape of the model body. Report – coding and circuit design section, begin creating the presentation.	Sai	02/05/2025
Rest of report, voiceover of the presentation.	Komal	02/05/2025
	Date of next meeting: 02/05/2025	

Ulster University **EEE416 Team Meeting Minutes**		
Design Team: Sai Adharsh Babu, Komal Bairwa		**Date: 02/05/2025**
Brief Note of Discussion: Tested the complete model, including the coding, circuit board and cardboard model in lab again. It sort of worked. Recorded the demo video.		
Actions	**Team member(s)**	**Deadline**

	responsible	
Record and edit the demo video. Complete report – coding and circuit design.	Sai	05/05/2025
Complete the report (add all the meeting minutes) and presentation video.	Komal	05/05/2025
	Date of next meeting: 06/05/2025	

Ulster University **EEE416 Team Meeting Minutes**		
Design Team: Sai Adharsh Babu, Komal Bairwa	**Date: 05/05/2025**	
Brief Note of Discussion: Upload the report, presentation, demo video and coding zip file via submission link on blackboard.		
Actions	**Team member(s) responsible**	**Deadline**
Upload the report, presentation, demo video and coding zip file via submission link on blackboard.	Sai	06/05/2025
Upload the report, presentation, demo video and coding zip file via submission link on blackboard.	Komal	06/05/2025
	Date of next meeting: None	