MACHINE LEARNING BASED SIGN LANGUAGE INTERPRETER

A Project Report Submitted in partial fulfilment of the Requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

VIYYURI PADMAJA (20021A0449)

BONDILI SANDEEP SINGH (20021A0405)

LAVUDI SAI ROHITH (20021A0447)

DADICHILUKA SHIVALALITH (20021A0438)

Under the esteemed guidance of

Smt. P. PUSHPALATHA

Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

KAKINADA – 533003 (A.P)

2020-2024

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

CERTIFICATE

This is to certify that the project entitled **"MACHINE LEARNING BASED SIGN LANGUAGE INTERPRETER"** that is being submitted by **V. Padmaja (20021A0449), B. Sandeep Singh (20021A0405), L. Sai Rohith (20021A0447), D. Shivalalith (20021A0438)** in partial fulfilment of the requirements for the award of the degree of the **Bachelor of Technology** in **Electronics and Communication Engineering** to the **University College of Engineering Kakinada, Jawaharlal Nehru Technological University Kakinada** is a record of bona fide work carried out by them under my guidance and supervision.

The results embodied in this project have not been submitted to any other university or institute for the award of any degree or diploma.

Project Guide
Smt. P. PUSHPALATHA,
Assistant Professor of ECE,
UCEK,
JNT University Kakinada,
Kakinada-533003, A.P.

Head of the Department
Dr. B. LEELA KUMARI,
Professor and HOD,
ECE Dept., UCEK,
JNT University Kakinada,
Kakinada-533003, A.P.

DECLARATION

We hereby declare that the work embodied in this project entitled **"MACHINE LEARNING BASED SIGN LANGUAGE INTERPRETER"**, which is being submitted by us in the requirement for the award of the degree of the **BACHELOR OF TECHNOLOGY** in **ELECTRONICS AND COMMUNICATION ENGINEERING** from Jawaharlal Nehru Technological University Kakinada (A.P), India, is the result of investigations carried out by us under the supervision of **Smt. P. PUSHPALATHA,** Assistant Professor in Electronics and Communication Engineering, University College of Engineering Kakinada, JNTU Kakinada.

The work is original, and the results of this project have not been submitted elsewhere for the award of any degree or diploma.

With gratitude,

VIYYURI PADMAJA (20021A0449)

BONDILI SANDEEP SINGH (20021A0405)

LAVUDI SAI ROHITH (20021A0447)

DADICHILUKA SHIVALALITH (20021A0438)

ACKNOWLEDGEMENT

We express our deep gratitude and regards to **Smt. P. PUSHPALATHA,** Project Guide and Assistant Professor, Department of Electronics and Communication Engineering. We are extremely thankful and indebted to her for sharing expertise, sincere and valuable guidance and encouragement extended to us.

We are pleased to thank **Dr. B. LEELA KUMARI,** Head of the Department, Department of Electronics and Communication Engineering, UCEK, JNTUK, Kakinada and **Dr. M. H. M. KRISHNA PRASAD**, Principal, UCEK, JNTUK, Kakinada who have rendered their constant encouragement and valuable suggestions in making the project successful.

We take this opportunity to thank all the **Professors and Faculty Members** in the department for their teachings and academic support and thanks to **Technical Staff and Non-Teaching Staff** in the department for their support.

With gratitude,

VIYYURI PADMAJA (20021A0449)

BONDILI SANDEEP SINGH (20021A0405)

LAVUDI SAI ROHITH (20021A0447)

DADICHILUKA SHIVALALITH (20021A0438)

CONTENTS

ABSTRACT

Communication between normal and handicapped people such as deaf, dumb, blind and paralyzed patients has always been a challenging task. However, sign language is the only way to communicate for listening and talking with disabled people. It has been observed that they find it very difficult at times to interact with normal people with their gestures, as only very few of those are recognized by the most people. We are developing a prototype to reduce the communication gap between differentially able and normal people.

Designing a smart hand glove makes sign language understandable to all. A wireless data gloves is used which is normal cloth driving gloves fitted with flex sensors along the length of each finger and the thumb. When mute people use these gloves to perform hand gestures, it will be converted into speech so that normal people can understand their expression.

In this project, flex sensor plays the major role. Flex sensors are sensors that change in resistance depending on the amount of bend on the sensor. Accelerometers can provide information about the orientation of the device. Further, the Machine Learning (ML) classifiers (LR, SVM, MLP and RF) are used for recognizing sign language. Then, utilizing a machine learning (ML) model serialized with the Pickle method to deploy a Flask API. Then, making use of an ESP32 microcontroller with Wi-Fi capability to transmit data from flex sensors and accelerometer to the API. Finally, display the API output within the Arduino IDE terminal and convert it into voice using commands.

Overall, this project aims to remove the communication barriers faced by those who rely on sign language and to improve their ability to communicate effectively in emergency situations.

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

Humans have used sign language as a means of communication and message delivery, particularly for the deaf people. To convey a speaker’s thoughts using sign language, hand shapes, hand orientation and movement, and facial expressions may all be used simultaneously. However, hardly many everyday individuals are conversant with sign language. Therefore, those who use sign language as a regular form of communication may find it challenging to converse with others or simply to express their views to others.

Therefore, as a result of the rapid advancement of technology, tools have been developed to enable deaf communities to hear or speak with others. There are several varieties of over-the-counter hearing aids available to help with deafness and other communication disorders, including behind-the-ear, in-the-ear, and canal aids. Even while hearing aids are useful, using one of these devices may cause the user to feel uncomfortable or to hear background noise. As a result, scientists have been working on numerous techniques that can translate sign language gestures. The two main techniques are wearable technology and vision-based systems. Vision-based systems use feature extraction techniques in image processing to determine the movements of the hands and fingers. The wearable will be coupled to a microcontroller in order to do away with cords, and the smart glove will use a variety of methods to interpret sign language into written or spoken words.

We designed and built a glove to be worn on the right hand that uses a Machine Learning (ML) algorithm to translate sign language into spoken English. Every person's hand is a unique size and shape, and we aimed to create a device that could provide reliable translations regardless of those differences. Our device uses five flex sensors that we use to quantify how much each finger is bent, and the MPU-6050 (a three-axis accelerometer and gyroscope) is able to detect the orientation and rotational movement of the hand. These sensors are read, averaged, and arranged into packets using ESP32 microcontroller. These packets are then sent serially to a user's PC to be run in conjunction with a Python script. The user creates data sets of information from the glove for each gesture that should eventually be translated, and the algorithm trains over these datasets to predict later at runtime what a user is signing.

Our goal is to create a way for the speech-impaired to be able to communicate with the public more easily. With a variety of sensors, we can quantify the state of the right hand in a series of numerical data. By collecting a moderate amount of this data for each letter or word and feeding it into a Machine Learning algorithm, it can train over this dataset and learn to associate a given hand gesture with its corresponding sign. We use Flex Sensors as variable resistors to detect how much each finger is bent, and the MPU-6050 can identify the orientation and hand movements that the Flex sensors cannot capture. Moreover, to discern between extremely similar signs, we add contact sensors to the glove to gather additional information.

1.2 LITERATURE SURVEY

S. Sundaram et al. proposed a tactile knitted glove assembled with 548 resistive sensors to achieve the signature recognition of human grasp, showing a great potential of using a glove to realize highly accurate object manipulator. However, complicated system design and data processing may restrict their broadened applications in human machine interactions since they need many sensors to capture comprehensive information for human/robotic hands.

In numerous scenarios, glove-based HMIs can perfectly meet the needs of control even equipped with several sensors, which is so called minimalist design. Accordingly, S. Choi et al. proposed an intuitive control interface involving 3 resistive sensors distributed on 3 fingers of the glove to perform VR shooting game, indicating a simple HMI solution. Similarly, K. Suzuki et. al. used 9 carbon nanotube (CNT)-based resistive strain sensors as components of a textile-based glove for the real-time finger joint motion detection.

Kelly et al. (2009) devised a multimodal system employing HMMs, which concurrently recognizes hand gestures and head movements. Proposing a multimodal framework, Yang and Lee (2011) utilized Hierarchical Conditional Random Fields (H-CRF) and SVMs to discern hand gestures and facial expressions, respectively. Subsequently, they introduced a second multimodal framework with three cameras capturing various orientations (Yang & Lee, 2013). Chen, Tian, Liu, and Metaxas (2013) introduced a novel multimodal framework amalgamating face and body gestures. It utilizes Histogram of Oriented Gradients (HOG) features and an SVM classifier to identify ten distinct expressions.

In recent research by Huang, Zhou, Zhang, Li, and Li (2018), a multimodal framework for continuous sign language recognition was presented. Their approach incorporates global hand locations/motions and local hand gesture details, employing a Hierarchical Attention Network with Latent Space (LS-HAN). This framework demonstrated enhanced accuracy on the RWTH-PHOENIX-Weather 2014 benchmark dataset. Zhou, Zhou, et al. (2020) proposed a spatial–temporal multimodal network with joint optimization for end-to-end sequence learning.

In a recent study by Min, Hao, Chai, and Chen (2021), the iterative training approach utilized in recent works on Continuous Sign Language Recognition (CSLR) is examined using the RWTH-PHOENIX-Weather dataset.

1.3 OBJECTIVE

Design and integrate flex sensors, accelerometer, and ESP32 microcontroller to create a comprehensive sensory input system.

Employ machine learning techniques to process data from flex sensors and accelerometer embedded within the ESP32 platform.

Implement real-time speech and voice conversion algorithms on the ESP32 platform to translate sensory input into understandable speech output.

1.4 PROJECT ORGANIZATION

The entire project is divided into five chapters and is organized as

CHAPTER 1: Overview, a literature survey on sign language recognition, objectives of the project, and organization of the project.

CHAPTER 2: Introduction to multiple ways of sign language recognition, hardware components required and their description, software and hardware setup for interfacing with ESP32 microcontroller.

CHAPTER 3: Different ML classifiers, data description

CHAPTER 4: Model deployment

CHAPTER 5: Results

CHAPTER 6: Conclusion and future scope

CHAPTER-2
SMART GLOVE

2.1 INTRODUCTION

In the world's whole population, there are approximately 72 million deaf-mute people. To bridge the gap between the deaf and normal community, a smart glove is an excellent resort. Smart gloves can be implemented in either a vision-based or non-vision-based manner.

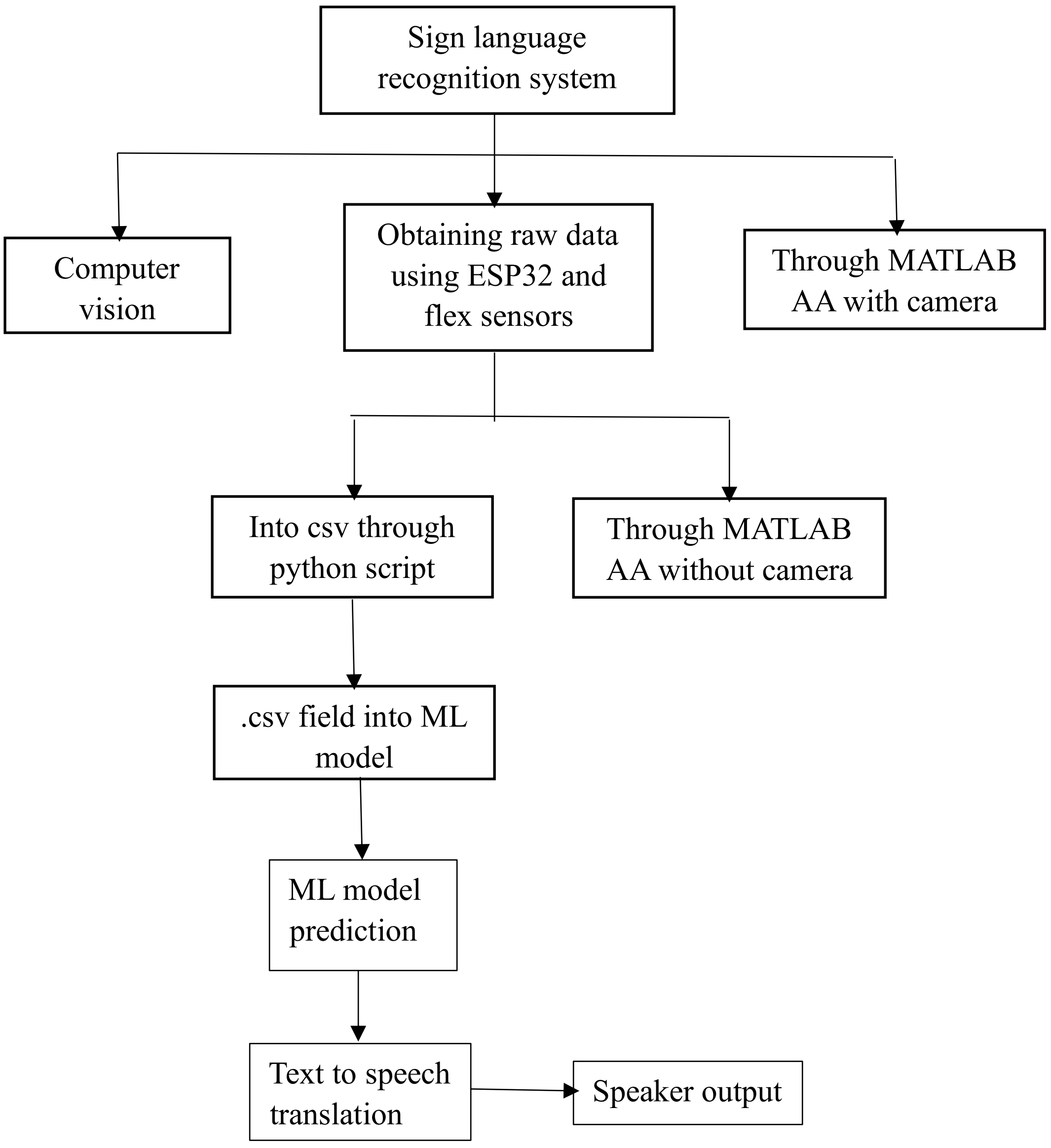

Sign language recognition system
Computer vision
Obtaining raw data using ESP32 and flex sensors
Through MATLAB AA with camera
Into csv through python script
Through MATLAB AA without camera
.csv field into ML model
ML model prediction
Text to speech translation
Speaker output

Fig. 2.1: "Flowchart for sign language recognition system"

In vision-based approach a camera reads the sign and uses deep learning algorithms to identify the sign and convert it to speech. However, this approach can be computationally complex and slower in converting the signs to speech.

Therefore, we propose to apply non-vision-based approach where sensors (usually, Flex sensors and Accelerometer) are used to gather the information about the signs and use traditional Machine Learning (ML) algorithms to estimate the sign and convert to speech.

NON-VISION-BASED APPROACH

Speech-impaired people need to communicate with the normal people in an easier manner and for this purpose a system was designed to produce proper sign outputs regardless of factors like the size and shape of the hand.

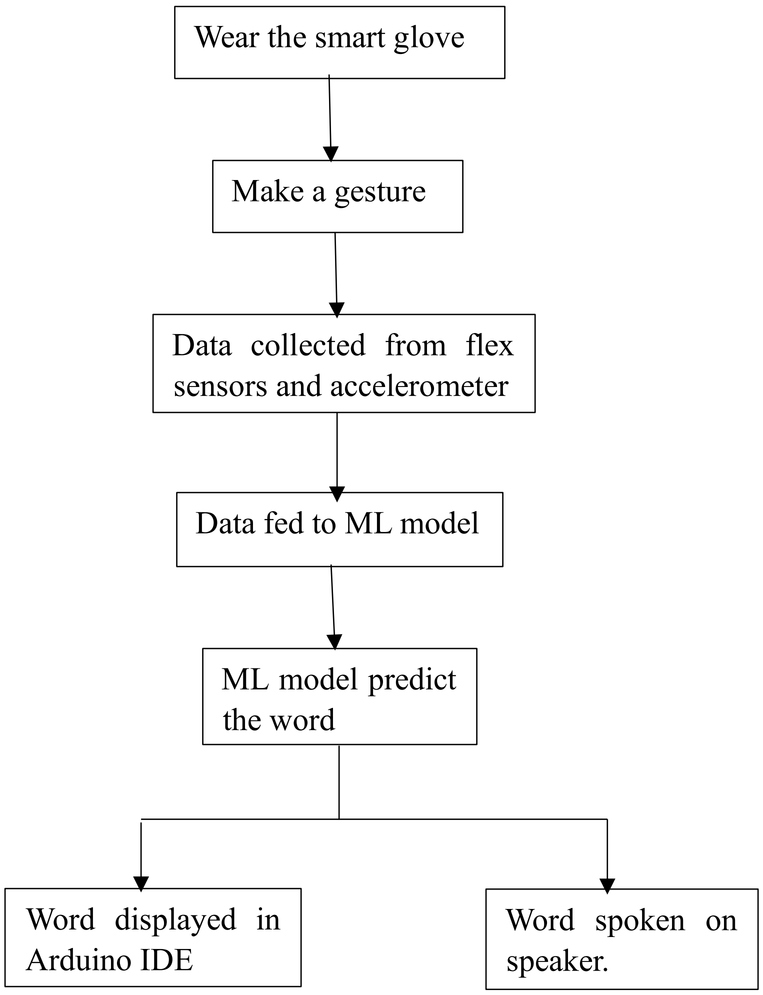

Wear the smart glove
Make a gesture
Data collected from flex sensors and accelerometer
Data fed to ML model
ML model predict the word
Word displayed in Arduino IDE
Word spoken on speaker.

Fig. 2.2: "Proposed smart glove approach using ML"

The glove uses the SVM ML algorithm to translate sign language to speech. The system makes use of five flex sensors, MPU6050 accelerometer, and ESP32 microcontroller.

Flex sensors were used to determine the bending angle of signs, accelerometer for the position/orientation of the sign. The data was read and then sent to the microcontroller, by the user's PC to run a python script to figure out the corresponding output. The output is then displayed in Arduino IDE and speech output is obatined through laptop speaker.

2.2 COMPONENTS USED

The hardware components used are

1. Five flex sensors
2. MPU-6050 accelerometer
3. ESP32 microcontroller
4. Breadboard
5. Connecting wires

FLEX SENSOR

A flex sensor is a type of sensor utilized for measuring deflection or bending. It is crafted using materials such as plastic and carbon. The carbon surface is applied onto a plastic strip, and when the strip bends, it alters the sensor's resistance. Consequently, it's often referred to as a bend sensor. Its resistance variation is directly linked to the degree of bending, making it useful as a goniometer.

PIN CONFIGURATION

The flex sensor's pin configuration is depicted below, featuring two terminals labelled as p1 and p2. Unlike diodes or capacitors, this sensor lacks polarized terminals, meaning there's no distinction between positive and negative. To activate the sensor, it requires a voltage ranging from 3.3V to 5V DC, which can be sourced from various interfacing methods.

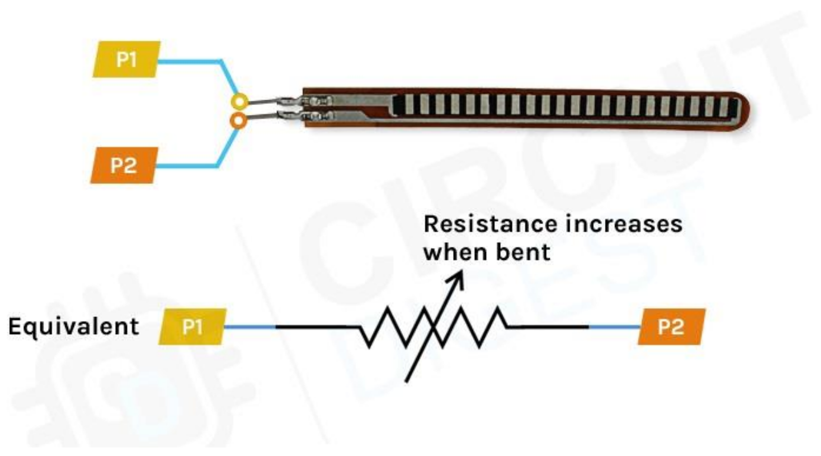


Fig. 2.3: "Flex sensor"

Pin P1: This pin is generally connected to the +ve terminal of the power source.
Pin P2: This pin is generally connected to GND pin of the power source.

WORKING PRINCIPLE

This sensor operates on the principle of a bending strip, meaning its resistance changes when the strip is twisted. This variation can be detected using any compatible controller. Functioning as a variable resistance, its resistance alters with twisting, influenced by the surface's linearity.

Fig. 2.4: "Flex sensor"

For instance, resistance differs when the sensor is level versus when it's twisted at 45° or 90°.

FLEX SENSOR FEATURES AND SPECIFICATIONS

- Operating voltage of FLEX SENSOR: 0-5V
- Can operate on LOW voltages.
- Power rating: 0.5Watt (continuous), 1 Watt (peak)
- Life: 1 million
- Operating temperature: -45ºC to +80ºC
- Flat Resistance: 25K Ω
- Resistance Tolerance: ±30%
- Bend Resistance Range: 45K to 125K Ohms (depending on bend)

HOW TO USE FLEX SENSOR

The flex sensor functions as a variable resistor, with its terminal resistance escalating as it bends. Thus, the increase in resistance is contingent upon the surface's linearity, making it ideal for detecting linear changes.

Under linear conditions, the flex sensor maintains its nominal resistance. However, when bent at a 45º angle, the sensor's resistance doubles compared to its original state. When bent at a 90º angle, the resistance could increase up to four times the nominal resistance. This indicates that the resistance across the terminals escalates linearly with the angle of bend. Thus, the flex sensor effectively transforms flex angle into a measurable resistance parameter.

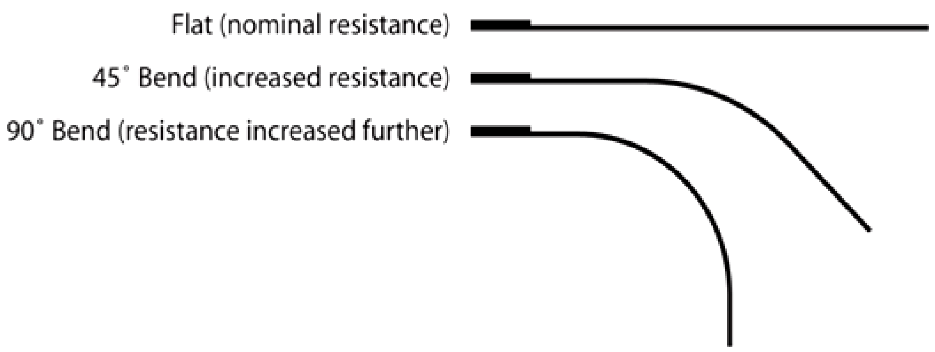

Flat (nominal resistance)
45˚ Bend (increased resistance)
90˚ Bend (resistance increased further)

Fig. 2.5: "Flex sensor bending angles"

For convenience we convert this resistance parameter to voltage parameter. For that we are going to use a voltage divider circuit.

In this resistive network we have two resistances. One is constant resistance (R1) and other is variable resistance (RV1).

Vo is the voltage at midpoint of voltage divider circuit and is also the output voltage. Vo is also the voltage across the variable resistance (RV1). So, when the resistance value of RV1 is changed the output voltage Vo also changes. So, we will have resistance change in voltage change with voltage divider circuit.

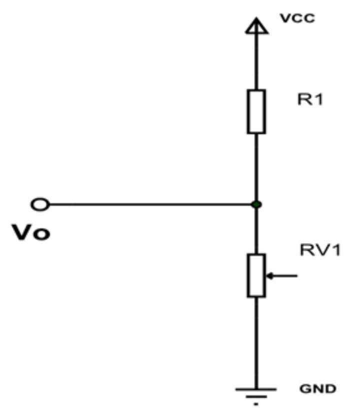


Fig. 2.6: "voltage divider circuit"

Here we will replace the variable resistance (RV1) with flex sensor.

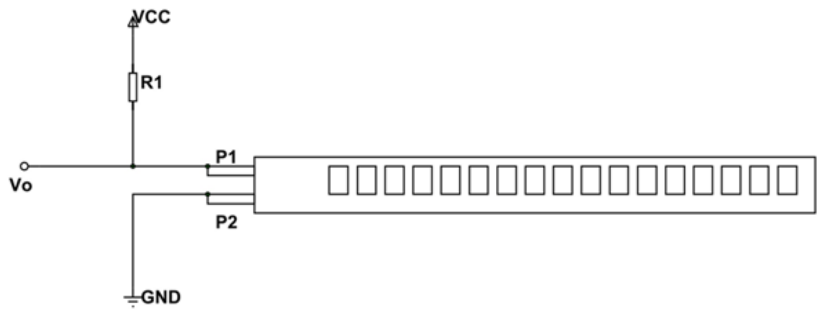


Fig. 2.7: "voltage divider in flex sensor connections"

As illustrated in the figure, R1 represents a constant resistance while the flex sensor serves as a variable resistance. Vo denotes the output voltage, corresponding to the voltage across the flex sensor.

In this setup,

Vo = VCC(Rx/(R1+Rx)),

where Rx denotes the flex sensor resistance.

When the flex sensor bends, its terminal resistance escalates, consequently affecting the voltage divider circuit. This leads to an increase in the voltage drop across the flex sensor and thus Vo. Consequently, as the flex sensor's bend increases, Vo voltage rises linearly, providing a voltage parameter indicative of the flex.

By utilizing this voltage parameter, we can input it into an ADC to obtain a digital value, which can be conveniently utilized.

APPLICATIONS

1. Gesture Recognition
2. Wearable Devices
3. Prosthetics and Robotics
4. Musical Instruments
5. Medical Devices
6. Industrial Automation
7. Human-Machine Interfaces
8. Navigation Systems
9. Security Systems

MPU6050 ACCELEROMETER AND GYROSCOPE SENSOR

The MPU6050 sensor module is a comprehensive 6-axis Motion Tracking Device, compactly integrating a 3-axis Gyroscope, 3-axis Accelerometer, and Digital Motion Processor. Additionally, it boasts an on-chip Temperature sensor for added functionality.

With an I2C bus interface, it facilitates seamless communication with microcontrollers. Furthermore, it features an Auxiliary I2C bus enabling communication with other sensor devices such as a 3-axis Magnetometer or Pressure sensor. When a 3-axis Magnetometer is connected to the auxiliary I2C bus, the MPU6050 can deliver a complete 9-axis Motion Fusion output, enhancing its versatility and utility.

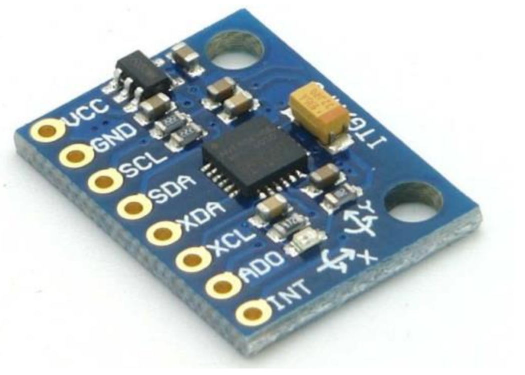


Fig. 2.8: "MPU6050 module"

The module includes an on-board LD3985 3.3V regulator, so you can safely use it with a 5V logic microcontroller like Arduino and ESP32.

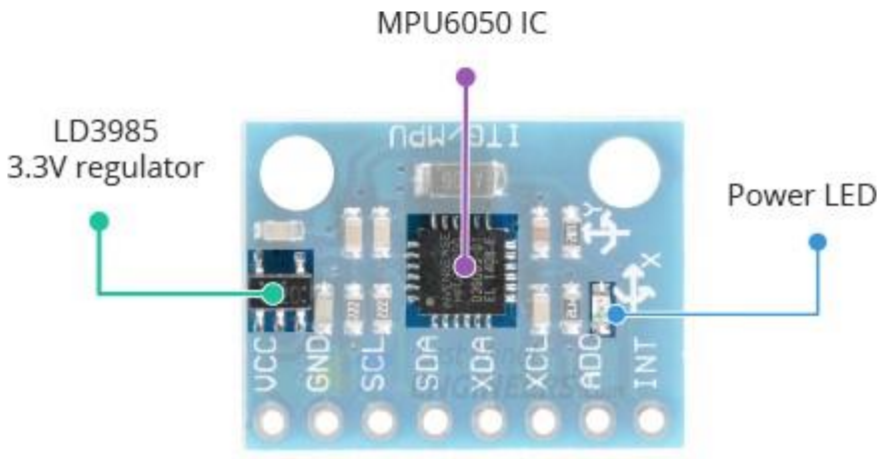


Fig. 2.9: "MPU6050 module hardware overview"

The MPU6050 consumes less than 3.6mA during measurements and only 5µA when idle. Because of its low power consumption, it can be used in battery-powered devices. Additionally, the module has a Power LED that illuminates when the module is powered on.

3-AXIS GYROSCOPE

The MPU6050 consist of 3-axis Gyroscope with Micro Electro Mechanical System (MEMS) technology. It is used to detect rotational velocity along the X, Y, Z axes as shown in the above figure. It measures the angular velocity along each axis in degree per second unit.

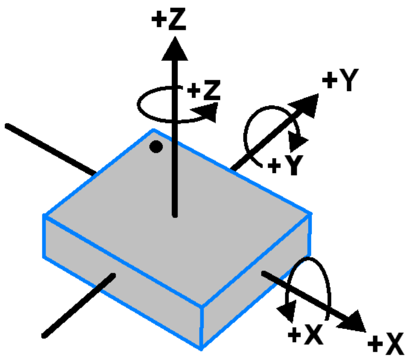


Fig. 2.10: "MPU6050 gyroscope orientation and polarity of rotation"

When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a vibration that is detected by a MEM inside MPU6050. The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate. This voltage is digitized using 16-bit ADC to sample each axis.

The full-scale range of output are +/- 250, +/- 500, +/- 1000, +/- 2000. It measures the angular velocity along each axis in degree per second unit.

The gyroscope measures rotational velocity or rate of change of the angular position over time, along the X, Y and Z axis. It uses MEMS technology and the Coriolis Effect for measuring. The outputs of the gyroscope are in degrees per second, so in order to get the angular position we just need to integrate the angular velocity.

3-AXIS ACCELEROMETER

The MPU6050 consists of 3-axis Accelerometer with Micro Electro Mechanical (MEMs) technology. It used to detect angle of tilt or inclination along the X, Y and Z axes as shown in below figure.

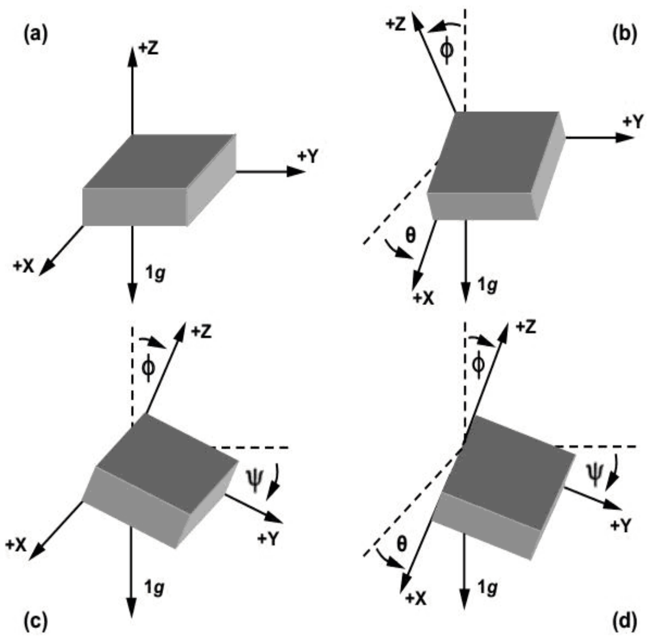


Fig. 2.11: "MPU6050 accelerometer axis"

Acceleration along the axes deflects the movable mass. This displacement of moving plate (mass) unbalances the differential capacitor which results in sensor output. Output amplitude is proportional to acceleration. 16-bit ADC is used to get digitized output.

The full-scale range of acceleration are +/- 2g, +/- 4g, +/- 8g, +/- 16g. It is measured in g (gravity force) unit. When device placed on flat surface it will measure 0g on X and Y axis and +1g on Z axis.

MPU6050 MODULE PINOUT

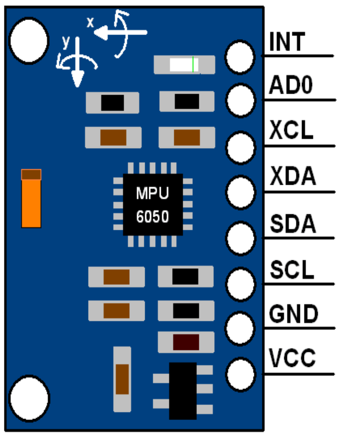


Fig. 2.12: “MPU6050 module pinout”

MPU6050 PIN DESCRIPTION

The MPU6050 module has 8 pins,

1. INT: Interrupt digital output pin.
2. AD0: I2C Slave Address LSB pin. This is 0th bit in 7-bit slave address of device. If connected to VCC then it is read as logic one and slave address changes.
3. XCL: Auxiliary Serial Clock pin. This pin is used to connect other I2C interface enabled sensors SCL pin to MPU-6050.
4. XDA: Auxiliary Serial Data pin. This pin is used to connect other I2C interface enabled sensors SDA pin to MPU-6050.
5. SCL: Serial Clock pin. Connect this pin to microcontrollers SCL pin.
6. SDA: Serial Data pin. Connect this pin to microcontrollers SDA pin.
7. GND: Ground pin. Connect this pin to ground connection.
8. VCC: Power supply pin. Connect this pin to +5V DC supply.

MPU-6050 module has Slave address (When AD0 = 0, i.e. it is not connected to Vcc) as, Slave Write address (SLA+W): 0xD0, Slave Read address (SLA+R): 0xD1.

SPECIFICATIONS OF MPU6050 SENSOR

GYROSCOPE

- 3-axis sensing with a full-scale range of ±250, ±500, ±1000, or ±2000 degrees per second (dps)
- Sensitivity of 131, 65.5, 32.8, or 16.4 LSBs per dps
- Output data rate (ODR) range of 8kHz to 1.25Hz

ACCELEROMETER

- 3-axis sensing with a full-scale range of ±2g, ±4g, ±8g, or ±16g
- Sensitivity of 16384, 8192, 4096, or 2048 LSBs per g
- ODR range of 8kHz to 1.25Hz
- Operating range of -40°C to +85°C
- Sensitivity of 340 LSBs per degree Celsius
- Accuracy of ±3°C

SUPPLY VOLTAGE

Operating voltage range of 2.375V to 3.46V for the MPU-6050, and 2.375V to 5.5V for the MPU-6050A

COMMUNICATION INTERFACE

I2C serial interface with a maximum clock frequency of 400kHz

8-bit and 16-bit register access modes.

APPLICATIONS

1. Motion Sensing and Tracking

2. Gesture Recognition and Human-Computer Interaction (HCI)

3. Vehicle Tracking and Navigation

4. Motion analysis and Biomechanics

ESP32 MICRO CONTROLLER

ESP32 comes with an on-chip 32-bit microcontroller with integrated Wi-Fi + Bluetooth + BLE features that targets a wide range of applications. It is a series of low-power and low-cost developed by Espressif Systems.

The original ESP32 chip had a single core Tensilica Xtensa LX6 microprocessor.

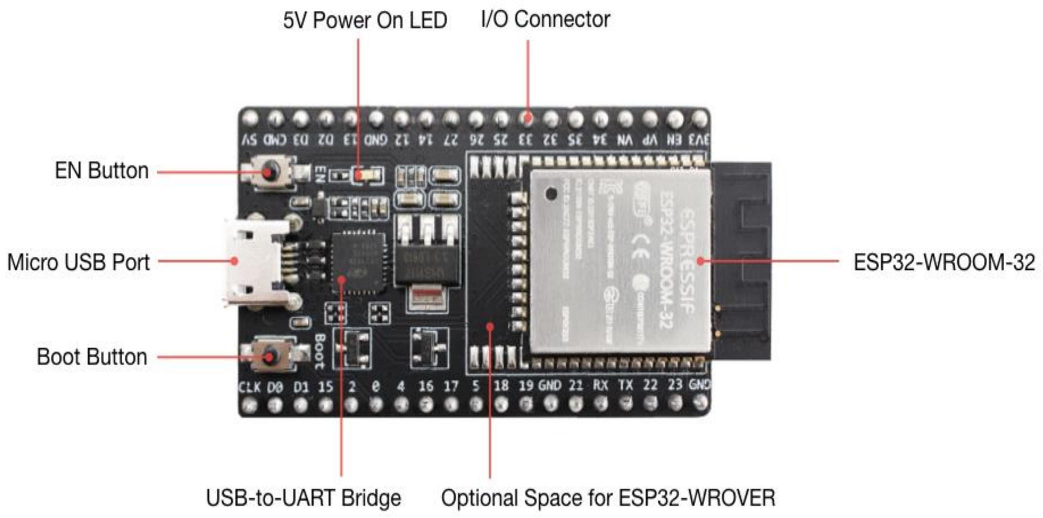


Fig. 2.13: "ESP32 microcontroller"

The processor had a clock rate of over 240 MHz, which made for a relatively high data processing speed. One of the standout features of the ESP32 is its built-in Wi-Fi (802.11 b/g/n) and Bluetooth (Bluetooth v4.2 BR/EDR and BLE). This connectivity enables IoT (Internet of Things) applications, remote monitoring, data logging, and wireless communication with other devices. The ESP32 offers a wide array of peripherals including GPIO, SPI, I2C, UART, ADC, DAC, PWM, touch sensors, and more. These peripherals facilitate interfacing with various sensors, actuators, displays, and communication modules.

The ESP32 includes advanced power management features such as multiple sleep modes and power domains. Low-power capabilities make it suitable for battery-operated and energy-efficient applications. Developers can program the ESP32 using various development environments including Arduino IDE, ESP-IDF (Espressif IoT Development Framework), MicroPython, and others. The ESP32 includes hardware-based security features such as Secure Boot, Flash Encryption, and cryptographic accelerators. These features help protect against unauthorized access, data breaches, and tampering.

The ESP32 offers a cost-effective solution for IoT and embedded projects, providing a balance between performance, features, and affordability. The ESP32 is highly scalable, with variations available in terms of memory, peripheral configuration, and form factors. The newer models are available with combined Wi-Fi and Bluetooth connectivity, or just Wi-Fi connectivity. There are several different chip models available, including:

- ESP32-D0WDQ6 (and ESP32D0WD)
- ESP32-D2WD
- ESP32-S0WD
- System in package (SiP) – ESP32-PICO-D4
- ESP32 S series
- ESP32-C series
- ESP32-H series

ESP32	**DESCRIPTION**
Core	2
Architecture	32 bits
Clock	Tensilica Xtensa LX106 160-240 MHz
Wi-Fi	IEEE802.11 b/g/n
Bluetooth	Yes - classic and BLE
RAM	520 KB
Flash	External QSP – 16 MB
GPIO	22
DAC	2
ADC	18
Interfaces	SPI-I2C-UART-I2S-CAN

Fig. 2.14: “Specifications of ESP32”

PROCESSOR: The ESP32 employs a Tensilica Xtensa 32-bit LX6 microprocessor, typically featuring a dual-core architecture, except for the ESP32-S0WD module, which utilizes a single-core system. With clock frequencies reaching up to 240MHz, it delivers performance of up to 600 DMIPS (Dhrystone millions of instructions per second). Its efficient power management enables analog-to-digital conversions, computations, and level thresholds even in deep sleep mode.

WIRELESS CONNECTIVITY: The ESP32 provides integrated Wi-Fi connectivity compliant with 802.11 b/g/n/e/i standards. Additionally, it supports Bluetooth connectivity with v4.2 BR/EDR and Bluetooth Low Energy (BLE) capabilities.

MEMORY: The internal memory configuration of the ESP32 includes ROM (448 KB for booting/core functions), SRAM (520 KB for data/instructions), RTC fast SRAM (8 KB for data storage/main CPU during boot from sleep mode), RTC slow SRAM (8 KB for co-processor access during sleep mode), and eFuse (1 KiBit, with 256 bits utilized for system functions such as MAC address and chip configuration, and 768 bits reserved for customer applications).

Some ESP32 chips like the ESP32-D2WD and ESP32-PICO-D4 feature internally connected flash memory. Please refer to the respective internal flash memory specifications for each ESP32 chip variant.

EXTERNAL FLASH AND SRAM: The ESP32 supports up to four 16 MB external QSPI flashes and SRAMs, with hardware encryption based on AES to safeguard developers' programs and data. It accesses external QSPI flash and SRAM via high-speed caches.

ESP32 FUNCTIONS

ESP32 has many applications when it comes to IoT.

Here are some of the IoT functions the chip is used for.

NETWORKING: The module's Wi-Fi antenna and dual core enables embedded devices to connect to routers and transmit data.

DATA PROCESSING: The ESP32 processor is versatile, capable of handling basic inputs from analog and digital sensors, as well as executing more intricate calculations with either a Real-Time Operating System (RTOS) or a non-OS Software Development Kit (SDK). A non-OS SDK operates directly on the chip without the need for a full operating system to support it.

P2P CONNECTIVITY: Creates direct communication between different ESPs and other devices using IoT P2P connectivity.

WEB SERVER: Provides access to pages written in HTML or development languages.

APPLICATIONS

1. Internet of Things (IoT) Devices
2. Home automation
3. Robotics
4. Wearables
5. Audio and noise applications
6. Industrial monitoring and control
7. Environmental monitoring
8. Smart lighting
9. Security systems

BREADBOARD

A breadboard is a rectangular plastic board with a bunch of tiny holes in it. These holes let you easily insert electronic components to prototype electronic circuit, like resistor, capacitor, transistor and an LED (light-emitting diode).

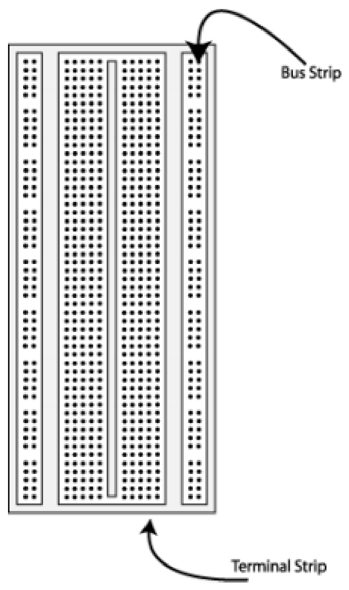


Fig. 2.15: "Breadboard"

The connections are not permanent, so it is easy to remove a component if you make a mistake, or just start over and do a new project.

There are long rows of holes called strips. Each breadboard has two types of strips, bus strips and terminal strips. Bus strips let you connect the board and its electronic components to a power source. Terminal strips let you plug various electronic components in and connect them to each other.

Inside, metal strips create connections between the electronic components that you plug in. These connections create an electronic circuit, which you can then use to control any of your electronics projects.

MALE TO FEMALE JUMPER WIRES

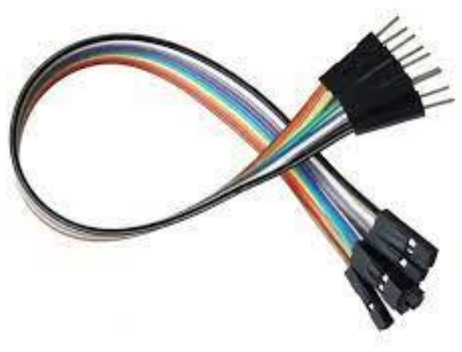


Fig. 2.16: “male to female jumper wires”

These are male to female jumper wires used in connecting the female header pin of any development board to other development boards having a male connector. They are simple wires that have connector pins at each end allowing them to be used to connect two points to each other.

We used these wires for connections with flex sensors.

MALE TO MALE JUMPER WIRES

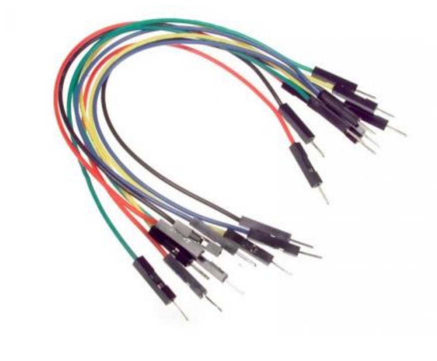


Fig. 2.17: “male to male jumper wires”

Male-to-male jumper wires are the most common and what you likely will use most often. When connecting two ports on a breadboard, a male-to-male wire is used. It is used to interconnect the components of a breadboard or other prototype or test circuit internally or with other equipment or components without soldering.

2.3 INTERFACING ESP32 WITH FLEX SENSORS AND MPU6050

We need to interface ESP32 microcontroller with 5 flex sensors and an accelerometer to get values for different hand gestures made for our sign language recognition.

In order to get those values we need to connect those components using connecting wires and a bread board. And then the required code should be written in Arduino IDE can get us the required results.

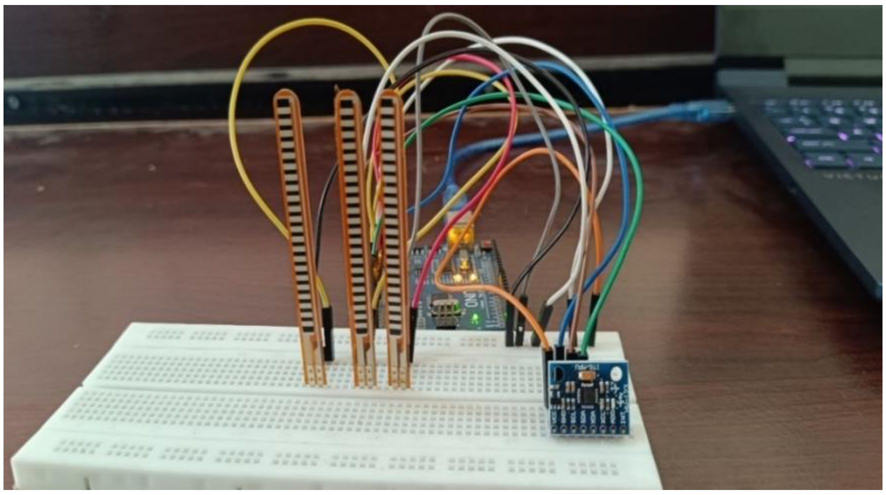


Fig. 2.18: "Interfacing ESP32 with flex sensors and MPU6050"

ARDUINO CODE

```
#include <Wire.h>
#include <MPU6050.h>
#define FLEX_SENSOR_PIN_1 32 // Example GPIO pin for flex sensor 1
#define FLEX_SENSOR_PIN_2 33 // Example GPIO pin for flex sensor 2
#define FLEX_SENSOR_PIN_3 25 // Example GPIO pin for flex sensor 3
#define FLEX_SENSOR_PIN_4 26 // Example GPIO pin for flex sensor 4
#define FLEX_SENSOR_PIN_5 27 // Example GPIO pin for flex sensor 5

#define MPU6050_SDA 21 // Example GPIO pin for MPU6050 SDA
#define MPU6050_SCL 22 // Example GPIO pin for MPU6050 SCL
```

```
MPU6050 mpu(Wire);
void setup() {
Serial.begin(9600);
 // Initialize MPU6050
Wire.begin(MPU6050_SDA, MPU6050_SCL);
 mpu.begin();
  // Set up flex sensor pins
  pinMode(FLEX_SENSOR_PIN_1, INPUT);
  pinMode(FLEX_SENSOR_PIN_2, INPUT);
  pinMode(FLEX_SENSOR_PIN_3, INPUT);
  pinMode(FLEX_SENSOR_PIN_4, INPUT);
  pinMode(FLEX_SENSOR_PIN_5, INPUT);
}
void loop() {
  // Read flex sensor values
  int flex1 = analogRead(FLEX_SENSOR_PIN_1);
   int flex2 = analogRead(FLEX_SENSOR_PIN_2);
  int flex3 = analogRead(FLEX_SENSOR_PIN_3);
  int flex4 = analogRead(FLEX_SENSOR_PIN_4);
  int flex5 = analogRead(FLEX_SENSOR_PIN_5);
  // Read accelerometer data
  int16_t ax, ay, az;
  mpu.getAcceleration(&ax, &ay, &az);
  // Print sensor data
  Serial.print("Flex Sensors: ");
  Serial.print(flex1);
  Serial.print(", ");
  Serial.print(flex2);
  Serial.print(", ");
  Serial.print(flex3);
  Serial.print(", ");
  Serial.print(flex4);
  Serial.print(", ");
  Serial.println(flex5);
  Serial.print("Accelerometer: ");
  Serial.print(ax);
  Serial.print(", ");
  Serial.print(ay);
  Serial.print(", ");
  Serial.println(az);
  delay(1000);
}
```

2.4 HARDWARE SETUP

The interfacing of ESP32 with 5 flex sensors and MPU6050 accelerometer can be performed by following the below connections.

FLEX SENSOR

Connect one terminal of the flex sensor to GND and the other terminal to a fixed resistor (e.g., 10kΩ). Connect the other end of the fixed resistor 3.3V Connect the junction between the flex sensor and the resistor to a GPIO pin on the ESP32 (e.g., GPIO34).

Repeat this setup for each flex sensor, using different GPIO pins on the ESP32 for each sensor.

This setup creates a voltage divider circuit, where the voltage at the junction between the flex sensor and the resistor varies based on the flex sensor's resistance (which changes with bending). By reading the voltage at this junction using the ESP32's analog-to-digital converter (ADC), you can determine the degree of flex.

Ensure that the GPIO pins used for flex sensors are defined correctly in your code to match the physical connections.

MPU6050 ACCELEROMETER

Connect the VCC pin of the MPU6050 to 3.3V or 5V output of the ESP32 GND pin of the MPU6050 to GND pin of the ESP32. Connect the SDA (Serial Data) pin of the MPU6050 to a designated GPIO pin on the ESP32 (e.g., GPIO21). Connect the SCL (Serial Clock) pin of the MPU6050 to another GPIO pin on the ESP32 (e.g., GPIO22).

Make sure to double-check all connections and ensure they match the pin mappings defined in your code. Once you've made the connections, you can proceed with uploading and running the code on the ESP32 to read data from the flex sensors and MPU6050 accelerometer

2.5 PROCEDURE

To obtain the data from flex sensors and accelerometer we need to follow the below procedure.

1. Arduino IDE:

   Download and install the Arduino IDE

2. ESP32 Board Installation:

   Go to "Tools" > "Board" > "Boards Manager...".

   Search for "ESP32" and install the ESP32 platform by Espressif Systems.

   Select your ESP32 board from "Tools" > "Board".

3. Libraries Installation:

   Install the required libraries for the MPU6050 accelerometer and Wire communication. For MPU6050, install the "MPU6050" library by Jeff Rowberg using the Arduino Library Manager. For Wire, this is a built-in library, so no installation is needed.

4. Writing the Code:

   Open a new sketch in the Arduino IDE.

   Copy and paste the code provided earlier into the sketch.

5. Upload the Code:

Connect your ESP32 to your computer via USB.

Select the appropriate COM port from "Tools" > "Port".

Click the "Upload" button in the Arduino IDE to compile and then upload the code to your ESP32.

6. Testing:

Open the Serial Monitor in the Arduino IDE to view the sensor readings.

Bend the flex sensors and observe changes in the readings.

Tilt or move the MPU6050 module to observe changes in accelerometer data.

OUTPUT

```
Flex Sensors: 512, 423, 345, 678, 789

Accelerometer: -123, 456, 789

Flex Sensors: 510, 425, 348, 680, 790

Accelerometer: -124, 457, 788

Flex Sensors: 508, 427, 347, 681, 791

Accelerometer: -125, 458, 787
```

"Flex Sensors" display the analog readings from each of the five flex sensors connected to the ESP32.

"Accelerometer" displays the raw accelerometer data (X, Y, Z) from the MPU6050 accelerometer.

These values will vary based on the actual readings from the sensors and the physical conditions (e.g., bending of flex sensors, orientation of the accelerometer).

CHAPTER 3
MACHINE LEARNING (ML) CLASSIFIERS

3.1 MACHINE LEARNING (ML) CLASSIFIERS

Machine learning (ML) algorithms are computational techniques that enable computers to learn from data and make predictions or decisions without being explicitly programmed.

Here are some ML algorithms and their potential use in our project:

1. Support Vector Machine (SVM)
2. Multi-Layered Perceptron (MLP)
3. Random Forest (RF)
4. Logistic Regression (LR)

SUPPORT VECTOR MACHINE (SVM)

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane.

WORKING OF SVM

Linear Separability: SVM works best when the data is linearly separable, meaning there exists a hyperplane that can completely separate the classes. However, it can also be extended to handle non-linearly separable data using kernel tricks.

Margin Maximization: SVM aims to maximize the margin, which is the distance between the hyperplane and the nearest data points (support vectors) from each class. Maximizing the margin helps in achieving better generalization and robustness.

Support Vectors: Support vectors are the data points that lie closest to the decision boundary (hyperplane). They are the critical points that determine the position and orientation of the hyperplane.

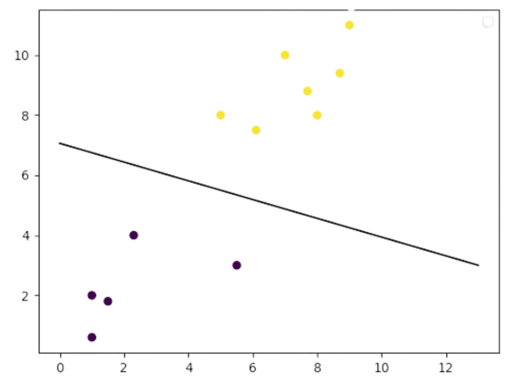


Fig. 3.1: "Linear SVM"

Kernel Trick: SVM can be extended to handle non-linearly separable data by using kernel functions to map the input features into a higher-dimensional space where the classes become separable. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid kernels.

Regularization Parameter(C): The regularization parameter(C) controls the trade-off between maximizing the margin and minimizing the classification error. A higher value of C allows for a smaller margin but fewer misclassifications, while a lower value of C prioritizes a larger margin even if it leads to more misclassifications.

Soft Margin SVM: In cases where the data is not perfectly separable, SVM can be extended to use a soft margin, allowing for some misclassifications. The balance between margin maximization and classification error is controlled by the regularization parameter (C).

To use SVM in Python with scikit-learn, the steps to be followed are

1. Import the SVM module:

```python
   from sklearn.svm import SVC
```

2. Create an SVM classifier object:

```python
svm_classifier = SVC(kernel='linear', C=1.0)
```

3. Train the classifier on your training data:

```python
svm_classifier.fit(X_train, y_train)
```

4. Make predictions on your test data:

```python
y_pred = svm_classifier.predict(X_test)
```

5. Evaluate the performance of the classifier:

```python
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
```

MULTI-LAYERED PERCEPTRON (MLP)

Multi-Layered perceptron defines the most complex architecture of artificial neural networks. An Artificial Neural Network is a training model with interconnected layers. ANN is the field of Artificial Intelligence in which a model is trained with training data, and it is expected that model will behave accurately with the actual data set. Multilayer Perceptron (MLP) is a type of artificial neural network (ANN) characterized by multiple layers of interconnected neurons (nodes).

A multilayer perceptron is a type of feedforward neural network consisting of fully connected neurons with a nonlinear kind of activation function. It is widely used to distinguish data that is not linearly separable.

Artificial Neural Network has three layers:

1. Input Layer
2. Hidden Layer
3. Output Layer

The pictorial representation of multi-layer perceptron learning is as shown below-

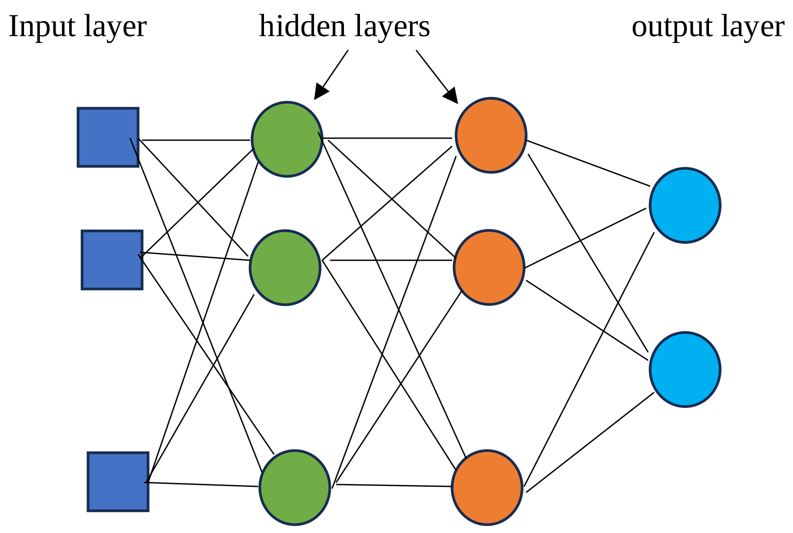


Fig. 3.2: “Multi-Layered Perceptron (MLP) layers”

Input layer: This layer consists of nodes or neurons that receive the initial input data. Each neuron represents a feature or dimension of the input data. The number of neurons in the input layer is determined by the dimensionality of the input data.

Hidden layer: Between the input and output layers, there can be one or more layers of neurons. Each neuron in a hidden layer receives inputs from all neurons in the previous layer (either the input layer or another hidden layer) and produces an output that is passed to the next layer.

Output layer: This layer consists of neurons that produce the final output of the network. The number of neurons in the output layer depends on the nature of the task.

MLPs are trained using the backpropagation algorithm, which computes gradients of a loss function with respect to the model's parameters and updates the parameters iteratively to minimize the loss.

WORKING OF MLP

Architecture: MLP consists of an input layer, one or more hidden layers, and an output layer. Each layer contains multiple neurons that are interconnected with neurons in adjacent layers. The hidden layers allow the network to learn hierarchical representations of the input data.

Feedforward Propagation: During feedforward propagation, input data is passed through the network layer by layer, and each neuron computes a weighted sum of its inputs, applies an activation function, and passes the result to the next layer. This process continues until the output layer produces a prediction.

Activation Functions: Activation functions introduce non-linearity into the network, allowing it to learn complex mappings between inputs and outputs. Common activation functions include sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax.

Backpropagation: Backpropagation is the process of updating the network's weights and biases based on the error between the predicted output and the true output. This process involves calculating the gradient of the loss function with respect to the network parameters and using gradient descent or a related optimization algorithm to minimize the loss.

Training: MLPs are trained using labeled data (supervised learning) through an iterative optimization process. The training data is passed through the network multiple times (epochs), and the network's parameters are adjusted to minimize the loss function.

Hyperparameters: Hyperparameters such as the number of layers, the number of neurons per layer, the choice of activation functions, and the learning rate play a crucial role in determining the performance of the MLP model.

To use MLP in Python with scikit-learn, the steps to be followed are

1. Import the MLP module:

~~~
```python

from sklearn.neural_network import MLPClassifier
~~~

2. Create an MLP classifier object:

```python
mlp_classifier =MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000, random_state=42)
```

Here, `hidden_layer_sizes` specifies the number of neurons in each hidden layer, `max_iter` controls the maximum number of iterations for training, and `random_state` ensures reproducibility of results.

3. Train the classifier on your training data:

```python
mlp_classifier.fit(X_train, y_train)
```

4. Make predictions on your test data:

```python
y_pred = mlp_classifier.predict(X_test)
```

5. Evaluate the performance of the classifier:

```python
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)
```

RANDOM FOREST (RF)

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."

Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

Random Forest algorithm is a powerful tree learning technique in Machine Learning. It works by creating a number of Decision Trees during the training phase. Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition. This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance.

In prediction, the algorithm aggregates the results of all trees, either by voting (for classification tasks) or by averaging (for regression tasks) This collaborative decision-making process, supported by multiple trees with their insights, provides an example stable and precise results. Random forests are widely used for classification and regression functions, which are known for their ability to handle complex data, reduce overfitting, and provide reliable forecasts in different environments.

The below diagram explains the working of the Random Forest algorithm.

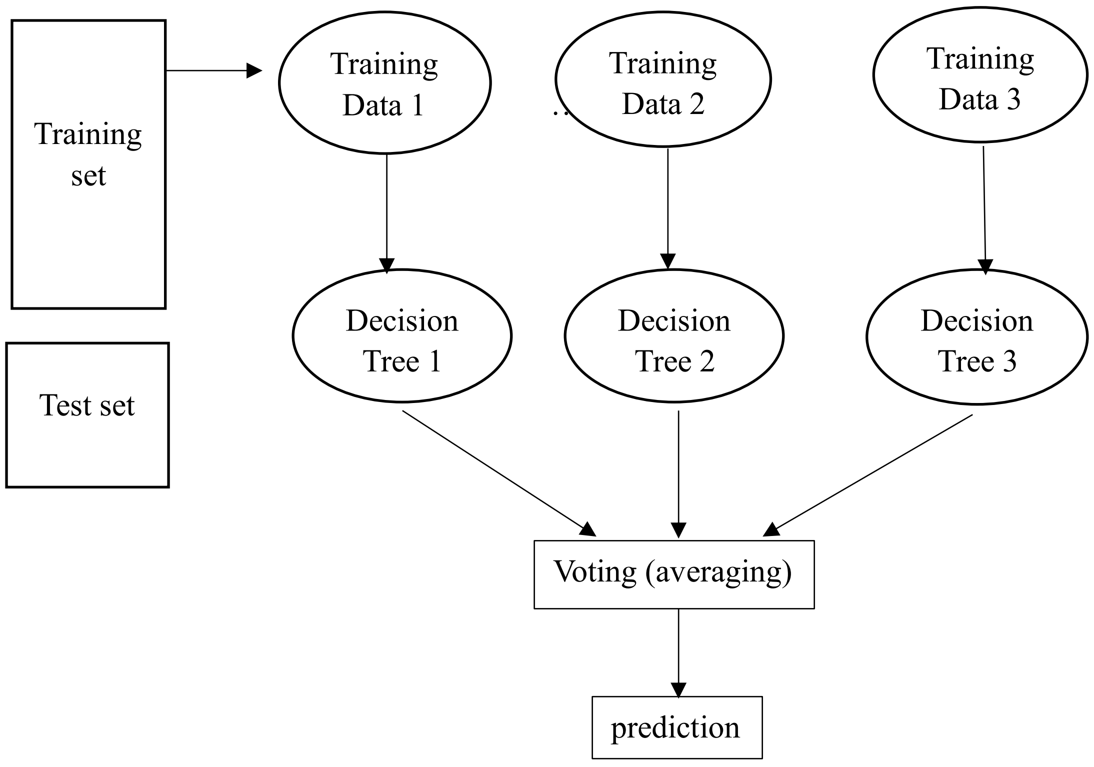


Fig. 3.3: "Random Forest (RF)"

WORKING OF RF

Decision Trees: Random Forest consists of a collection of decision trees, where each tree is trained independently on a random subset of the training data and features. Decision trees partition the feature space into regions and make predictions based on the majority class (for classification) or the average value (for regression) of the training samples in each region.

Random Subsampling: Random Forest randomly selects a subset of the training data (with replacement) and a subset of features at each node of the decision tree. This randomness helps to decorrelate the individual trees and reduce the variance of the ensemble model.

Bootstrap Aggregating (Bagging): Random Forest employs a technique called bagging, which involves training multiple decision trees on different subsets of the data and averaging their predictions.

Voting or Averaging: For classification tasks, Random Forest aggregates the predictions of individual trees through majority voting. For regression tasks, Random Forest aggregates the predictions by averaging the outputs of individual trees.

Hyperparameters: Random Forest has hyperparameters that control its behavior, including the number of trees (n_estimators), the maximum depth of each tree (max_depth), the minimum number of samples required to split a node (min_samples_split), and the maximum number of features to consider when looking for the best split (max_features).

To use Random Forest in Python with scikit-learn, the steps to be followed are:

1. Import the Random Forest module:

```python
from sklearn.ensemble import RandomForestClassifier  # For classification tasks

# from sklearn.ensemble import RandomForestRegressor  # For regression tasks
```

2. Create a Random Forest classifier object:

```python
```

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42) # Specify the number of trees (e.g., 100)

# rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42) # For regression tasks
```

3. Train the classifier on your training data:

```python
rf_classifier.fit(X_train, y_train)
```

4. Make predictions on your test data:

```python
y_pred = rf_classifier.predict(X_test)
```

5. Evaluate the performance of the classifier:

```python
from sklearn.metrics import accuracy_score  # For classification tasks

# from sklearn.metrics import mean_squared_error  # For regression tasks

accuracy = accuracy_score(y_test, y_pred)  # For classification tasks

# mse = mean_squared_error(y_test, y_pred)  # For regression tasks
```

LOGISTIC REGRESSION (LR)

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.

In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.

The below image is showing the logistic function

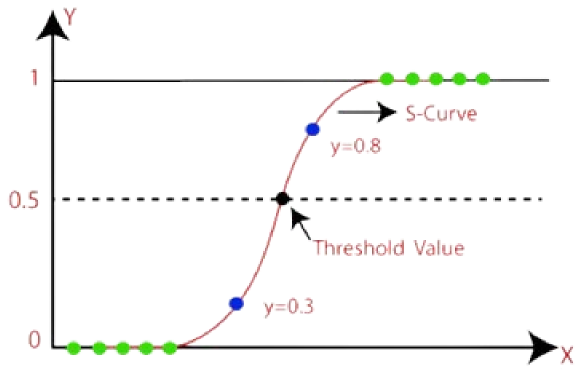


Fig. 3.4: "Logistic Regression (LR)"

WORKING OF LR

Sigmoid Function: Logistic Regression models the probability of the positive class (class 1) using the sigmoid function (also known as the logistic function), which ensures that the predicted probabilities are bounded between 0 and 1.

The sigmoid function is defined as

sigma(z) = 1/ (1 + e^{-z})

where z is the linear combination of input features and coefficients.

Model Representation: Logistic Regression models the log-odds (logit) of the probability of the positive class using a linear equation:

log [p/(1-p)] = B0 + B1 . x_1 + B2 . x_2 + ……. + Bn . x_n where p is the probability of the positive class, (x_1, x_2, ….. x_n) are the input features, (B0, B1, …. Bn) are the coefficients, and log represents the natural logarithm.

Loss Function: Logistic Regression minimizes the cross-entropy loss (log loss) between the predicted probabilities and the true labels. The log loss function penalizes incorrect predictions more strongly when the predicted probability diverges from the true label.instance is classified as belonging to the positive class; otherwise, it is classified as belonging to the negative class.

To use Logistic Regression in Python with scikit-learn, the steps to be followed are:

1. Import the Logistic Regression module:

```python
from sklearn.linear_model import LogisticRegression
```

2. Create a Logistic Regression model object:

```python
logistic_model = LogisticRegression()
```

3. Train the model on your training data:

```python
logistic_model.fit(X_train, y_train)
```

4. Make predictions on your test data:

```python
y_pred = logistic_model.predict(X_test)
```

5. Evaluate the performance of the model:

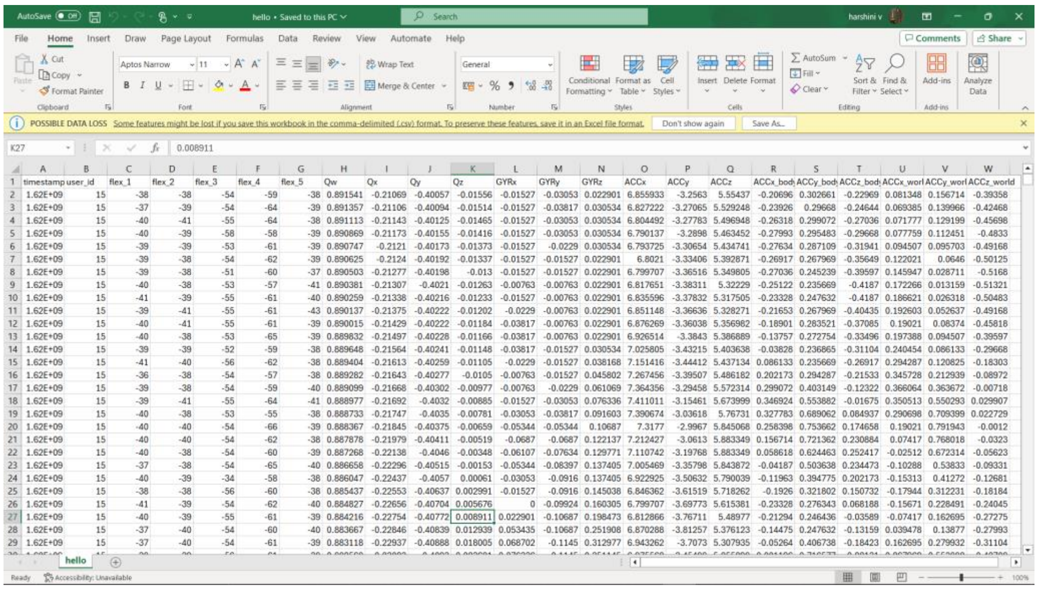
```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

classification_rep = classification_report(y_test, y_pred)
```

3.2 DATASET DESCRIPTION

The process of data collection, pre-processing, training, and testing process are described in dataset description.

DATA COLLECTION

Since ML relies heavily on the amount and quality of data, thus collecting the data is a crucial step in our approach. It is important to ensure that the data is a true representation of the system variables and has a large quantity, since the classification accuracy of the ML algorithms will heavily depend on it.


POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

K27 0.008911

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	timestamp	user_id	flex_1	flex_2	flex_3	flex_4	flex_5	Qw	Qx	Qy	Qz	GYRx	GYRy	GYRz	ACCx	ACCy	ACCz	ACCx_bod	ACCy_bod	ACCz_bod	ACCx_worl	ACCy_worl	ACCz_world
2	1.62E+09	15	-38	-38	-54	-59	-38	0.891541	-0.21069	-0.40057	-0.01556	-0.01527	-0.03053	0.022901	6.855933	-3.2563	5.55437	-0.20696	0.302661	-0.22969	0.081348	0.156714	-0.39358
3	1.62E+09	15	-37	-39	-54	-64	-39	0.891357	-0.21106	-0.40094	-0.01514	-0.01527	-0.03817	0.030534	6.827222	-3.27065	5.529248	-0.23926	0.29668	-0.24644	0.069385	0.139966	-0.42468
4	1.62E+09	15	-40	-41	-55	-64	-38	0.891113	-0.21143	-0.40125	-0.01465	-0.01527	-0.03053	0.030534	6.804492	-3.27783	5.496948	-0.26318	0.299072	-0.27036	0.071777	0.129199	-0.45698
5	1.62E+09	15	-40	-39	-58	-58	-39	0.890869	-0.21173	-0.40155	-0.01416	-0.01527	-0.03053	0.030534	6.790137	-3.2898	5.463452	-0.27993	0.295483	-0.29668	0.077759	0.112451	-0.4833
6	1.62E+09	15	-39	-39	-53	-61	-39	0.890747	-0.2121	-0.40173	-0.01373	-0.01527	-0.0229	0.030534	6.793725	-3.30654	5.434741	-0.27634	0.287109	-0.31941	0.094507	0.095703	-0.49168
7	1.62E+09	15	-39	-38	-54	-62	-39	0.890625	-0.2124	-0.40192	-0.01337	-0.01527	-0.01527	0.022901	6.8021	-3.33406	5.392871	-0.26917	0.267969	-0.35649	0.122021	0.0646	-0.50125
8	1.62E+09	15	-39	-38	-51	-60	-37	0.890503	-0.21277	-0.40198	-0.013	-0.01527	-0.01527	0.022901	6.799707	-3.36516	5.349805	-0.27036	0.245239	-0.39597	0.145947	0.028711	-0.5168
9	1.62E+09	15	-40	-38	-53	-57	-41	0.890381	-0.21307	-0.4021	-0.01263	-0.00763	-0.00763	0.022901	6.817651	-3.38311	5.32229	-0.25122	0.235669	-0.4187	0.172266	0.013159	-0.51321
10	1.62E+09	15	-41	-39	-55	-61	-40	0.890259	-0.21338	-0.40216	-0.01233	-0.01527	-0.00763	0.022901	6.835596	-3.37832	5.317505	-0.23328	0.247632	-0.4187	0.186621	0.026318	-0.50483
11	1.62E+09	15	-39	-41	-55	-61	-43	0.890137	-0.21375	-0.40222	-0.01202	-0.0229	-0.00763	0.022901	6.851148	-3.36636	5.328271	-0.21653	0.267969	-0.40435	0.192603	0.052637	-0.49168
12	1.62E+09	15	-40	-41	-55	-61	-39	0.890015	-0.21429	-0.40222	-0.01184	-0.03817	-0.00763	0.022901	6.876269	-3.36038	5.356982	-0.18901	0.283521	-0.37085	0.19021	0.08374	-0.45818
13	1.62E+09	15	-40	-38	-53	-65	-39	0.889832	-0.21497	-0.40228	-0.01166	-0.03817	-0.00763	0.022901	6.926514	-3.3843	5.386889	-0.13757	0.272754	-0.33496	0.197388	0.094507	-0.39597
14	1.62E+09	15	-39	-39	-52	-59	-38	0.889648	-0.21564	-0.40241	-0.01148	-0.03817	-0.01527	0.030534	7.025805	-3.43215	5.403638	-0.03828	0.236865	-0.31104	0.240454	0.086133	-0.29668
15	1.62E+09	15	-41	-40	-56	-62	-38	0.889404	-0.21613	-0.40259	-0.01105	-0.0229	-0.01527	0.038168	7.151416	-3.44412	5.437134	0.086133	0.235669	-0.26917	0.294287	0.120825	-0.18303
16	1.62E+09	15	-36	-38	-54	-57	-38	0.889282	-0.21643	-0.40277	-0.0105	-0.00763	-0.01527	0.045802	7.267456	-3.39507	5.486182	0.202173	0.294287	-0.21533	0.345728	0.212939	-0.08972
17	1.62E+09	15	-39	-38	-54	-59	-40	0.889099	-0.21668	-0.40302	-0.00977	-0.00763	-0.0229	0.061069	7.364356	-3.29458	5.572314	0.299072	0.403149	-0.12322	0.366064	0.363672	-0.00718
18	1.62E+09	15	-39	-41	-55	-64	-41	0.888977	-0.21692	-0.4032	-0.00885	-0.01527	-0.03053	0.076336	7.411011	-3.15461	5.673999	0.346924	0.553882	-0.01675	0.350513	0.550293	0.029907
19	1.62E+09	15	-40	-38	-53	-55	-38	0.888733	-0.21747	-0.4035	-0.00781	-0.03053	-0.03817	0.091603	7.390674	-3.03618	5.76731	0.327783	0.689062	0.084937	0.290698	0.709399	0.022729
20	1.62E+09	15	-40	-40	-54	-66	-39	0.888367	-0.21845	-0.40375	-0.00659	-0.05344	-0.05344	0.10687	7.3177	-2.9967	5.845068	0.258398	0.753662	0.174658	0.19021	0.791943	-0.0012
21	1.62E+09	15	-40	-40	-54	-62	-38	0.887878	-0.21979	-0.40411	-0.00519	-0.0687	-0.0687	0.122137	7.212427	-3.0613	5.883349	0.156714	0.721362	0.230884	0.07417	0.768018	-0.0323
22	1.62E+09	15	-40	-38	-54	-60	-39	0.887268	-0.22138	-0.4046	-0.00348	-0.06107	-0.07634	0.129771	7.110742	-3.19768	5.883349	0.058618	0.624463	0.252417	-0.02512	0.672314	-0.05623
23	1.62E+09	15	-37	-38	-54	-65	-40	0.886658	-0.22296	-0.40515	-0.00153	-0.05344	-0.08397	0.137405	7.005469	-3.35798	5.843872	-0.04187	0.503638	0.234473	-0.10288	0.53833	-0.09331
24	1.62E+09	15	-40	-39	-34	-58	-38	0.886047	-0.22437	-0.4057	0.00061	-0.03053	-0.0916	0.137405	6.922925	-3.50632	5.790039	-0.11963	0.394775	0.202173	-0.15313	0.41272	-0.12681
25	1.62E+09	15	-38	-38	-56	-60	-38	0.885437	-0.22553	-0.40637	0.002991	-0.01527	-0.0916	0.145038	6.846362	-3.61519	5.718262	-0.1926	0.321802	0.150732	-0.17944	0.312231	-0.18184
26	1.62E+09	15	-41	-39	-54	-62	-40	0.884827	-0.22656	-0.40704	0.005676	0	-0.09924	0.160305	6.799707	-3.69773	5.615381	-0.23328	0.276343	0.068188	-0.15671	0.228491	-0.24045
27	1.62E+09	15	-40	-39	-55	-61	-39	0.884216	-0.22754	-0.40772	0.008911	0.022901	-0.10687	0.198473	6.812866	-3.76711	5.48977	-0.21294	0.246436	-0.03589	-0.07417	0.162695	-0.27275
28	1.62E+09	15	-37	-40	-54	-60	-40	0.883667	-0.22846	-0.40839	0.012939	0.053435	-0.10687	0.251908	6.870288	-3.81257	5.376123	-0.14475	0.247632	-0.13159	0.039478	0.13877	-0.27993
29	1.62E+09	15	-37	-40	-54	-61	-39	0.883118	-0.22937	-0.40888	0.018005	0.068702	-0.1145	0.312977	6.943262	-3.7073	5.307935	-0.05264	0.406738	-0.18423	0.162695	0.279932	-0.31104

Fig. 3.5: “A sample of the data from the ASL-Sensor dataglove dataset for “hello” stored in a .csv file”

The dataset that we collected in were, consisted of 1500 instances for each alphabet from DU-ASL-DATA-GLOVE-DB. We are considering a total of 8 features which came from the flex sensors and accelerometer from each instance.

The 5 features from the flex sensors were termed as flex0, flex1, flex2, flex3, flex4.

The 3 features from the accelerometer were termed as ACCx, ACCy and ACCz.

DATA PRE-PROCESSING

In this step we converted the raw data that was collected, into understandable information as well as cleaning the data.

This is another critical stage in ML as it guarantees that the data has the least number of errors in its representation and also to account for the missing data.

Here we removed the other sensor data and only considered only 8 values.

File name was considered as a target in the ml model.

Here we also perform, data labelling and identifying the predicted features.

TRAINING AND TESTING SETS

After collection and labelling of the data is done, the next step is to train the model which is teaching the model how to predict an output based on a set of inputs.

For this purpose, the entire dataset was split into two parts, one for training which is 80% of the data (160 instance for training set) and the remaining 20% for testing (40 instance for testing set).

A Python code was implemented to read the data from the .csv files into the arrays using Numpy and Pandas open-source libraries.

CHAPTER 4
MODEL DEPLOYMENT

4.1 FLASK API

Model deployment in machine learning integrates a model into an existing production environment, enabling it to process inputs and generate outputs.

Here, we are deploying our ML model using Flask method.

Flask, a popular lightweight WSGI web application framework for Python, has gained traction as a preferred option for deploying machine learning models.

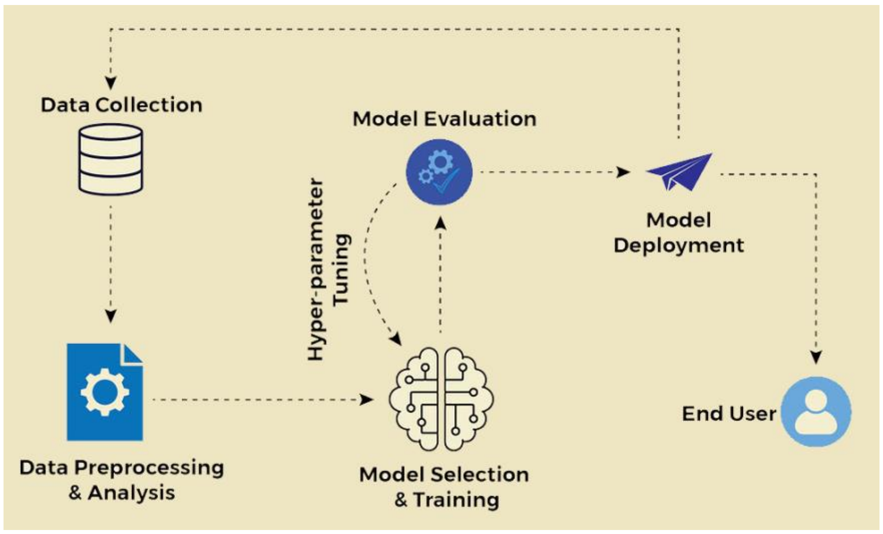


Fig. 4.1: “flowchart for final output”

PROCEDURE

1. Install Flask using pip: `pip install Flask`. Develop and train your machine learning model using libraries like TensorFlow, PyTorch, or scikit-learn. Save your trained model in a format that can be loaded by your Flask application
2. Create a Flask application: Import necessary modules: `Flask`, `request` from `flask`. Load your trained model into memory.
3. Define your Flask endpoints: Decide on the endpoints for your API (e.g., `/predict` for making predictions). Define the route handlers for each endpoint. For example, the `/predict` endpoint could accept POST requests containing data to be classified.

4. Handling data transfer using HTTP: In your Flask route handler functions, access incoming data using `request` object. You can receive data in different formats, such as JSON, form data, or file uploads. Process the incoming data as required by your machine learning model (e.g., image preprocessing, text tokenization). Make predictions using your trained model. Return the predictions or any relevant data as a response.

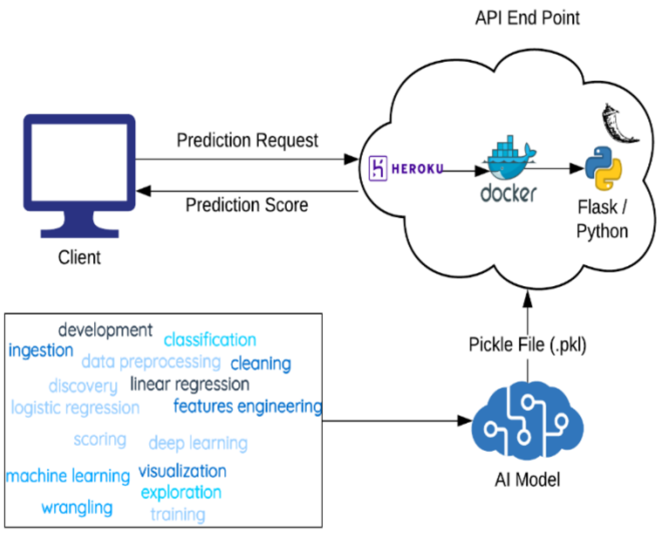


Fig. 4.2: "model deployment"

5. Sending data to the API: Data can be sent to the Flask API using various HTTP client libraries or tools. For example, you can use `requests` library in Python to send POST requests with data to your Flask API endpoint. Ensure that the data sent to the API matches the expected format and structure.
6. Handling responses: Once the Flask API processes the data and makes predictions, it sends back a response. The response can be in various formats, such as JSON, HTML, or plain text. Handle the response accordingly in your client application.
7. Testing: Test your Flask API endpoints locally to ensure they are functioning correctly. Use tools like Postman or cURL to send sample requests and verify the responses. Debug any issues and make necessary adjustments.
8. Deployment: Once your Flask application is ready, deploy it to a production server. Choose a suitable hosting platform (e.g., Heroku, AWS, Azure). Configure the server environment and deploy your Flask application. Ensure that the necessary dependencies and resources are available on the server.

9. Monitoring and maintenance: Monitor the performance of your deployed Flask API. Handle any errors or issues that arise in production. Regularly update and maintain your model and API as needed.

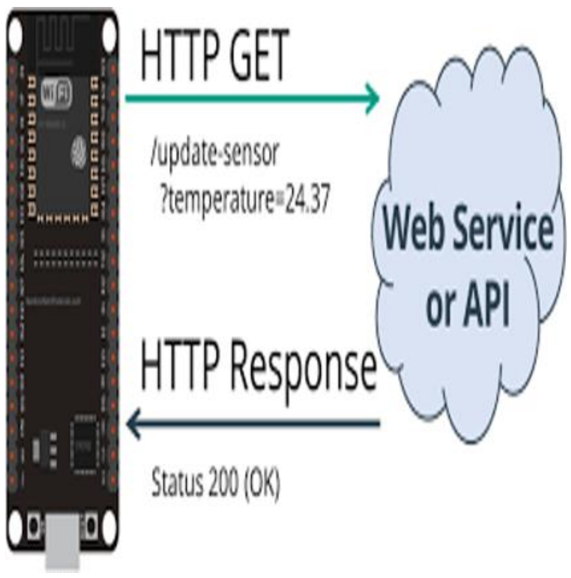


Fig. 4.3: "data from ESP32 to API and vice versa"

4.2 WAYS TO TRANSMIT DATA FROM ESP32 TO AN API

1. Wi-Fi:

ESP32 has built-in Wi-Fi capabilities, allowing it to connect to Wi-Fi networks and transmit data wirelessly. Wi-Fi enables communication with remote servers, cloud platforms, IoT devices, and other Wi-Fi-enabled devices. Wi-Fi offers high data transfer rates and long-range connectivity, making it ideal for applications requiring internet connectivity.

2. BLUETOOTH:

ESP32 supports Bluetooth communication through its built-in Bluetooth Low Energy (BLE) capabilities. BLE enables short-range wireless communication with smartphones, tablets, wearables, and other BLE devices. BLE is energy-efficient and suitable for applications such as wearable devices, health monitoring, and proximity sensing.

3. LoRa (Long Range):

ESP32 can communicate using LoRa technology, which enables long-range, low-power wireless communication. LoRa is ideal for applications requiring long-range communication, such as IoT deployments in remote areas, agriculture, environmental monitoring, and smart cities. However, LoRa typically has lower data transfer rates compared to Wi-Fi and Bluetooth.

4. ETHERNET:

ESP32 can be connected to a local network or the internet using Ethernet connectivity. Ethernet offers reliable and high-speed data transmission, making it suitable for applications requiring stable and fast communication. Ethernet is commonly used in industrial automation, smart buildings, and infrastructure projects.

Among all the possible ways, Wi-Fi is the best transmission method because of it’s high data transfer rates, wide coverage, internet connectivity, compatibility, scalability, security, and convenience.

4.3 TRANSMITTING DATA FROM ESP32 THROUGH Wi-Fi

To transmit data from a hand glove containing 5 flex sensors and an MPU6050 accelerometer to an API using the Wi-Fi capabilities of an ESP32 microcontroller, the steps to be followed are:

1. Set Up Wi-Fi Connection: Configure the ESP32 to connect to a Wi-Fi network. Provide the SSID and password of the Wi-Fi network using the ESP32's Wi-Fi library.
2. Collect Data: Collect the data that you want to transmit to the API. This could be sensor readings, device status, or any other relevant information.
3. Format Data: Format the collected data into a suitable format for transmission. This could be JSON, XML, or any other format depending on the requirements of the API.
4. Create HTTP Request: Create an HTTP request to send the data to the API. This typically involves specifying the HTTP method (e.g., POST), the API endpoint URL, headers (e.g., Content-Type), and the data payload.
5. Send HTTP Request: Use the ESP32's HTTP client library to send the HTTP request to the API endpoint over the Wi-Fi connection. Include the formatted data payload in the request body.
6. Handle Response: Handle the response from the API. This may involve parsing the response data, checking for errors or status codes, and taking appropriate action based on the response.
7. Error Handling: Implement error handling to deal with network connectivity issues, API errors, or other potential problems that may occur during the transmission process. Ensure that the data transmission process is secure by using HTTPS for encrypted communication, validating SSL certificates, and implementing authentication mechanisms if required by the API.
8. Security Considerations: Ensure that the data transmission process is secure by using HTTPS for encrypted communication, validating SSL certificates, and implementing authentication mechanisms if required by the API.
9. Testing and Debugging: Test the data transmission process thoroughly to ensure that it works as expected under different conditions. Use debugging tools and techniques to identify and resolve any issues that arise.

FLOWCHART

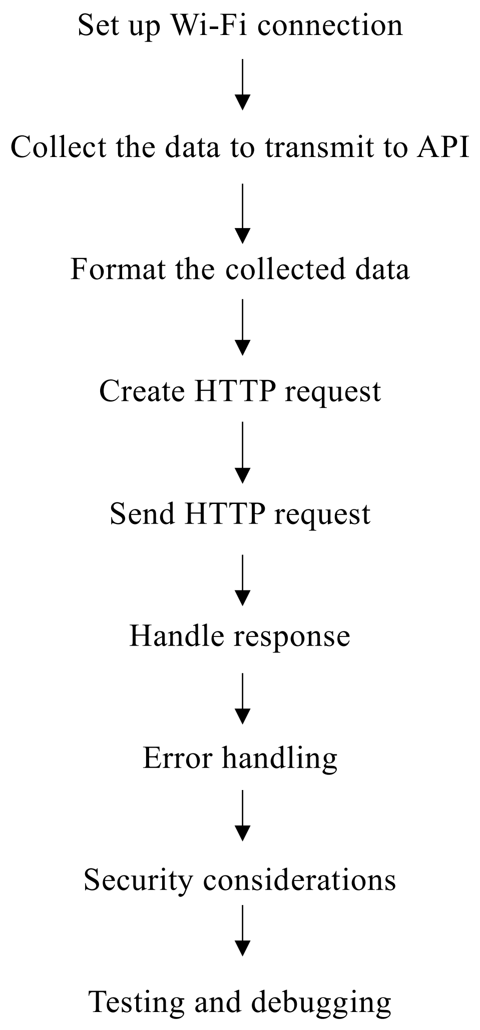


Fig. 4.4: "flowchart for transfer of data from ESP32 to API"

4.4 CODE FOR TRANSMITTING DATA THROUGH Wi-Fi FROM ESP32 TO API

```
#include <Wire.h>

#include <MPU6050.h>

#include <WiFi.h>

#include <HTTPClient.h>

#include <ArduinoJson.h>

#include "ESPAsyncWebServer.h"


#define WIFI_SSID "JNTUK_STAFF"

#define WIFI_PASSWORD "Jntuk@staffonly"

MPU6050 mpu

unsigned long sendDataPrevMillis = 0;

bool signupOK = true;

const int flexPin1 = 34;  // Analog input pin for Flex Sensor 1

const int flexPin2 = 35;  // Analog input pin for Flex Sensor 2

const int flexPin3 = 32;  // Analog input pin for Flex Sensor 3

const int flexPin4 = 33;  // Analog input pin for Flex Sensor 4

const int flexPin5 = 25;  // Analog input pin for Flex Sensor 5

AsyncWebServer server(80);

char serial[512];

void setup() {

Serial.begin(115200);

WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
```

```
Serial.print("Connecting to Wi-Fi");
while (WiFi.status() != WL_CONNECTED) {
Serial.print(".");
delay(300);
}
Serial.println();
Serial.print("Connected with IP: ");
Serial.println(WiFi.localIP());
Serial.println();
server.on("/data", HTTP_GET, [](AsyncWebServerRequest *request) {
request->send(200, "text/plain", serial);
});
Wire.begin(21, 22);  // SDA pin 21, SCL pin 22
mpu.initialize();
// Calibrate the accelerometer
mpu.CalibrateGyro();
}
void loop()
Serial.println(WiFi.localIP());
if (signupOK && (millis() - sendDataPrevMillis > 5000 || sendDataPrevMillis ==
0)) {
sendDataPrevMillis = millis();
// Read flex sensor values
float flexValue1 = analogRead(flexPin1);
float flexValue2 = analogRead(flexPin2);
float flexValue3 = analogRead(flexPin3);
float flexValue4 = analogRead(flexPin4);
float flexValue5 = analogRead(flexPin5);
// Read accelerometer data
int16_t ax, ay, az;
mpu.getAcceleration(&ax, &ay, &az);
WiFiClient wifiClient;
HTTPClient client;
client.begin(wifiClient, "http://192.168.29.121:5000/predict");
client.addHeader("Content-Type", "application/json");
const size_t capacity = JSON_OBJECT_SIZE(7);
StaticJsonDocument<capacity> doc;
doc["flex_1"] = flexValue1;
doc["flex_2"] = flexValue2;
doc["flex_3"] = flexValue3;
doc["flex_4"] = flexValue4;
doc["flex_5"] = flexValue5;
doc["ACCx"] = ax;
doc["ACCy"] = ay;
doc["ACCz"] = az;
String jsonStr;
serializeJson(doc, jsonStr);
int statusCode = client.POST(jsonStr);
if (statusCode > 0) {
```

```
Serial.print("HTTP status code: ");
Serial.println(statusCode);
String payload = client.getString();
Serial.println("Response: ");
Serial.println(payload);
const size_t capacity2 = JSON_OBJECT_SIZE(8);
StaticJsonDocument<capacity2> doc2;
doc2["flex_1"] = flexValue1;
doc2["flex_2"] = flexValue2;
doc2["flex_3"] = flexValue3;
doc2["flex_4"] = flexValue4;
doc2["flex_5"] = flexValue5;
doc2["ACCx"] = ax;
doc2["ACCy"] = ay;
doc2["ACCz"] = az;
doc2["message"] = payload;
serializeJson(doc2, Serial);
}
else {
Serial.print("Error: ");
Serial.println(statusCode);
} }
delay(10000);  // Add delay at the end of loop
}
```

CHAPTER 5

RESULTS

5.1 MODEL EVALUATION

Platform used: VS Code

Libraries used: Panda, NumPy, Scikit learn.

We have used SVM, MLP, RF, LR ML classifiers. The results for combined dataset for different ML classifiers are shown below. The results show the training and test accuracy and confusion matrix.

Support Vector Machine (SVM)

Results for combined dataset:

```
Training Accuracy (SVM): 0.9855208333333333

Test Accuracy (SVM): 0.9851666666666666

Confusion Matrix (SVM):

[[304   0   0 ...   1   0   0]
 [  0 305   0 ...   0   0   0]
 [  0   0 270 ...   0   0   0]
 ...
 [  0   0   0 ... 289   0   0]
 [  0   0   0 ...   0 262   2]
 [  0   0   0 ...   0   1 275]]
```

SVM classifier has training accuracy 98.55% and test accuracy 98.51%.

Multi-Layered Perceptron (MLP)

```
Results for combined dataset:

Training Accuracy (MLP): 0.996625

Test Accuracy (MLP): 0.9920833333333333

Confusion Matrix (MLP):

[[303   0   0 ...   1   0   0]
 [  0 305   0 ...   0   0   0]
 [  0   0 291 ...   0   0   0]
 ...
 [  0   0   0 ... 284   0   0]
 [  0   0   0 ...   0 283   3]
 [  0   0   0 ...   0   1 280]]
```

MLP classifier has training accuracy 99.66% and test accuracy 99.2%.

Random Forest (RF)

```
Results for combined dataset:

Training Accuracy (Random Forest): 1.0

Test Accuracy (Random Forest): 0.9978333333333333

Confusion Matrix (Random Forest):

[[305   0   0 ...   0   0   0]
 [  0 305   0 ...   0   0   0]
 [  0   0 300 ...   0   0   0]
 ...
 [  0   0   0 ... 289   0   0]
 [ 0   0   0 ...   0 295   0]
 [ 0   0   0 ...   0   0 282]]
```

RF classifier has training accuracy 100% and test accuracy 99.78%.

Logistic Regression (LR)

Results for combined dataset:

```
Training Accuracy (Logistic Regression): 0.9575
Test Accuracy (Logistic Regression): 0.9586666666666667
Confusion Matrix (Logistic Regression):
[[302   0   0 ...   1   0   0]
 [ 0 297   0 ...   0   0   0]
 [ 0   0 258 ...   0   0   0]
 ...
 [ 0   0   0 ... 273   0   0]
 [ 0   0   0 ...   0 243   4]
 [ 0   0   0 ...   0   1 269]]
```

LR classifier has training accuracy 95.75% and test accuracy 95.86%.

Among the four ML classifiers used (LR, SVM, MLP and RF), the RF classifier performed the best with test accuracy of 99.78% followed by MLP with 99.2%

5.2 OUTPUT

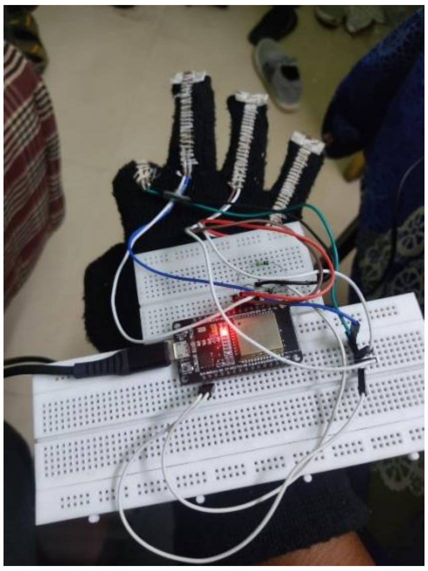
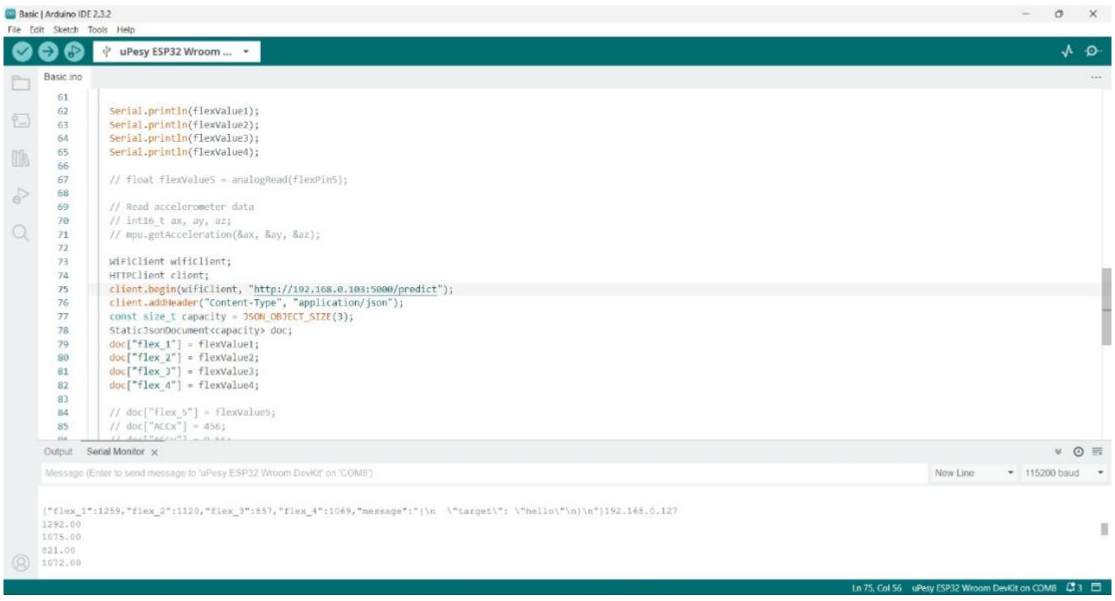


Fig. 5.1: “gesture 1 using smart glove”


Basic | Arduino IDE 2.3.2
File Edit Sketch Tools Help
uPesy ESP32 Wroom ...
Basic.ino
Output
Serial Monitor
Message (Enter to send message to 'uPesy ESP32 Wroom DevKit' on 'COM8')
New Line
115200 baud
Ln 75, Col 56 uPesy ESP32 Wroom DevKit on COM8

```
Serial.println(flexValue1);
Serial.println(flexValue2);
Serial.println(flexValue3);
Serial.println(flexValue4);

// float flexValue5 = analogRead(flexPin5);

// Read accelerometer data
// int16_t ax, ay, az;
// mpu.getAcceleration(&ax, &ay, &az);

WiFiClient wificlient;
HTTPClient client;
client.begin(wificlient, "http://192.168.0.103:5000/predict");
client.addHeader("Content-Type", "application/json");
const size_t capacity = JSON_OBJECT_SIZE(3);
StaticJsonDocument<capacity> doc;
doc["flex_1"] = flexValue1;
doc["flex_2"] = flexValue2;
doc["flex_3"] = flexValue3;
doc["flex_4"] = flexValue4;

// doc["flex_5"] = flexValue5;
// doc["ACCx"] = 456;
```

```
{"flex_1":1259,"flex_2":1120,"flex_3":857,"flex_4":1069,"message":"{\n  \"target\": \"hello\"\n}\n"}192.168.0.127
1292.00
1075.00
821.00
1072.00
```

Fig. 5.2: “output 1 in Arduino IDE”

When thumb finger is bent, according to our dataset of flex sensors values, we got the output “hello” in Arduino IDE terminal as well as speech output from laptop speaker.

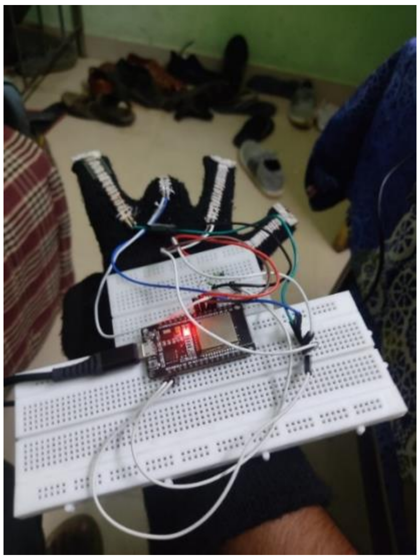
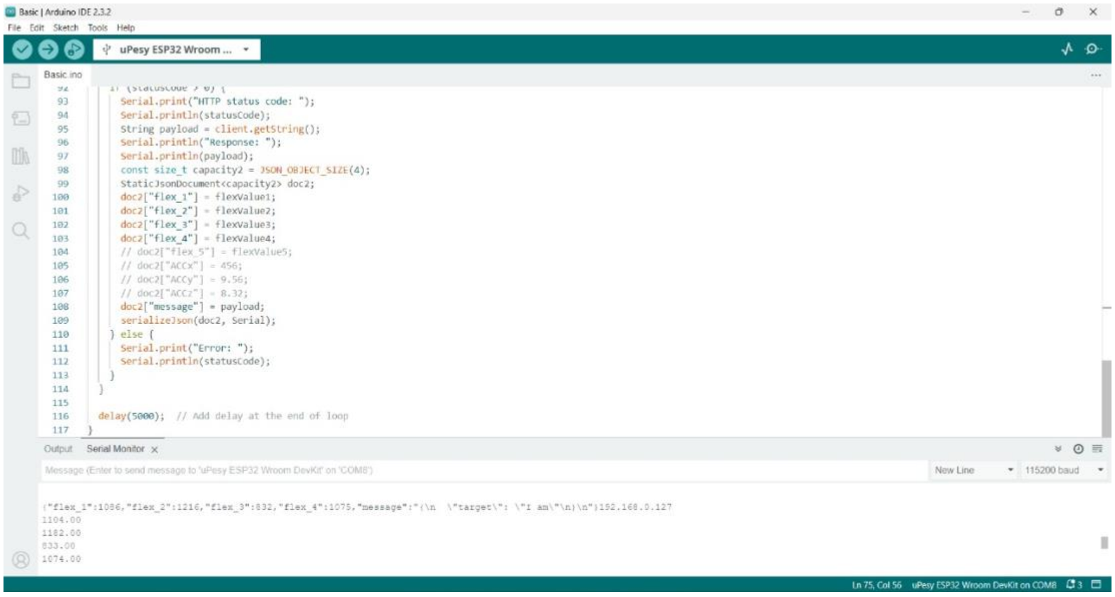


Fig. 5.3: “gesture 2 using smart glove”


Basic | Arduino IDE 2.3.2
File Edit Sketch Tools Help
uPesy ESP32 Wroom ...
Basic.ino
Output
Serial Monitor
New Line
115200 baud
Ln 75, Col 56 uPesy ESP32 Wroom DevKit on COM8

```
    if (statusCode > 0) {
      Serial.print("HTTP status code: ");
      Serial.println(statusCode);
      String payload = client.getString();
      Serial.println("Response: ");
      Serial.println(payload);
      const size_t capacity2 = JSON_OBJECT_SIZE(4);
      StaticJsonDocument<capacity2> doc2;
      doc2["flex_1"] = flexValue1;
      doc2["flex_2"] = flexValue2;
      doc2["flex_3"] = flexValue3;
      doc2["flex_4"] = flexValue4;
      // doc2["flex_5"] = flexValue5;
      // doc2["ACCx"] = 456;
      // doc2["ACCy"] = 9.56;
      // doc2["ACCz"] = 8.32;
      doc2["message"] = payload;
      serializeJson(doc2, Serial);
    } else {
      Serial.print("Error: ");
      Serial.println(statusCode);
    }
  }

  delay(5000);  // Add delay at the end of loop
}
```

```
{"flex_1":1086,"flex_2":1216,"flex_3":832,"flex_4":1075,"message":"{\n  \"target\": \"I am\"\n}\n"}192.168.0.127
1104.00
1182.00
833.00
1074.00
```

Fig. 5.4: “output 2 in Arduino IDE”

When thumb finger and middle finger both are bent at the same time, according to our dataset of flex sensors values, we got the output “I am” in Arduino IDE terminal as well as speech output from laptop speaker.

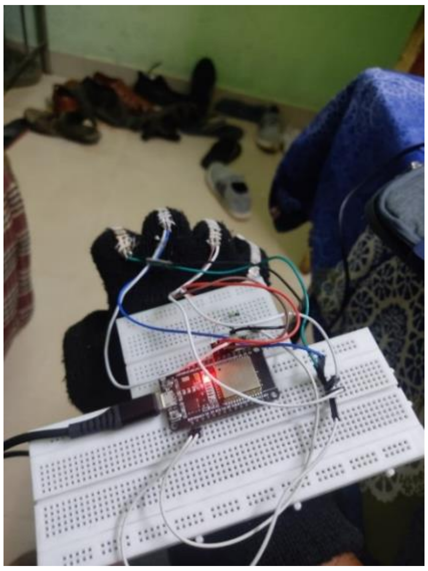
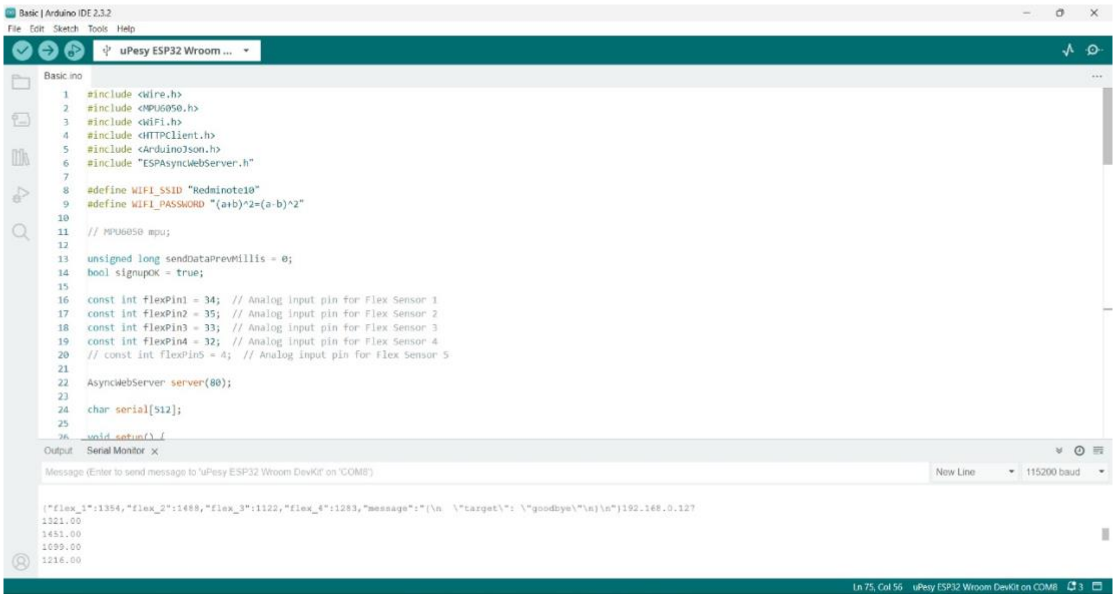


Fig. 5.5: “gesture 3 using smart glove”


Basic | Arduino IDE 2.3.2
File Edit Sketch Tools Help
uPesy ESP32 Wroom ...
Basic.ino
Output
Serial Monitor
Message (Enter to send message to 'uPesy ESP32 Wroom DevKit' on 'COM8')
New Line
115200 baud
Ln 75, Col 56 uPesy ESP32 Wroom DevKit on COM8

```
#include <Wire.h>
#include <MPU6050.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include "ESPAsyncWebServer.h"

#define WIFI_SSID "Redminote10"
#define WIFI_PASSWORD "(a+b)^2=(a-b)^2"

// MPU6050 mpu;

unsigned long sendDataPrevMillis = 0;
bool signupOK = true;

const int flexPin1 = 34;  // Analog input pin for Flex Sensor 1
const int flexPin2 = 35;  // Analog input pin for Flex Sensor 2
const int flexPin3 = 33;  // Analog input pin for Flex Sensor 3
const int flexPin4 = 32;  // Analog input pin for Flex Sensor 4
// const int flexPin5 = 4;  // Analog input pin for Flex Sensor 5

AsyncWebServer server(80);

char serial[512];

void setup() {
```

```
{"flex_1":1354,"flex_2":1488,"flex_3":1122,"flex_4":1283,"message":"{\n  \"target\": \"goodbye\"\n}\n"}192.168.0.127
1321.00
1451.00
1099.00
1216.00
```

Fig. 5.6: “output 3 in Arduino IDE”

When all the fingers are bent at the same time, according to our dataset of flex sensors values, we got the output “goodbye” in Arduino IDE terminal as well as speech output from laptop speaker.

CHAPTER 6
CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

The main aim of our project is to reduce the communication gap between mute and normal people. The hand glove (hardware) we developed involved ESP32 microcontroller, five flex sensors and MPU6050 accelerometer to provide sign language recognition and conversion of signs to speech and text. This converts the sign gesture made to text as well as speech. So, it is easy for differentially abled people to communicate with normal people.

We designed and implemented this approach using ML classifiers.

Displayed the corresponding output for the sign made using hand gloves in Arduino IDE terminal as well as speech output is obtained.

Different ML classifiers are used and compared.

Among the four ML classifiers (LR, SVM, MLP and RF), the RF classifier performed the best with a classification accuracy of 99.7% followed by MLP with 99.08%

6.2 FUTURE SCOPE

Hand glove consisting of flex sensors and ESP32 is used to translate the sign language and the output is shown in Arduino IDE terminal. We can also adapt the sign language interpreter for deployment on mobile devices or wearable platforms, enabling users to access sign language translation services on-the-go.

REFERENCES

[1] Ahmad Sami Al-Shamayleh, Rodina Ahmad, Mohammad AM Abushariah, Khubaib Amjad Alam, and Nazean Jomhari. A systematic literature review on vision based gesture recognition techniques. Multimedia Tools and Applications, 77(21):28121–28184, 2018.

[2] N Arun, R Vignesh, B Madhav, Arun Kumar, and S Sasikala. Flex sensor dataset: Towards enhancing the performance of sign language detection system. In 2022 International Conference on Computer Communication and Informatics (ICCCI), pages 01–05. IEEE, 2022.

[3] Bijay Sapkota, Mayank K Gurung, Prabhat Mali, and Rabin Gupta. Smart glove for sign language translation using arduino. In 1st KEC Conference Proceedings, volume 1, pages 5–11, 2018.

[4] Dhawal L Patel, Harshal S Tapase, Paraful A Landge, Parmeshwar P More, and AP Bagade. Smart hand gloves for disable people. Int. Res. J. Eng. Technol.(IRJET), 5:1423–1426, 2018.

[5] Firas A Raheem and Hadeer A Raheem. American sign language recognition using sensory glove and neural network. AL-yarmouk J, 11:171–182, 2019.

[6] Mina I Sadek, Michael N Mikhael, and Hala A Mansour. A new approach for designing a smart glove for arabic sign language recognition system based on the statistical analysis of the sign language. In 2017 34th National Radio Science Conference (NRSC), pages 380–388. IEEE, 2017.

[7] Mohammad A Alzubaidi, Mwaffaq Otoom, and Areen M Abu Rwaq. A novel assistive glove to convert arabic sign language into speech. Transactions on Asian and Low-Resource Language Information Processing, 2022.

[8] Tariq Jamil. Design and implementation of an intelligent system to translate arabic text into arabic sign language. In 2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), pages 1–4. IEEE, 2020.

[9] Teak-Wei Chong and Boon-Giin Lee. American sign language recognition using leap motion controller with machine learning approach. Sensors, 18(10):3554, 2018.

[10] Tushar Chouhan, Ankit Panse, Anvesh Kumar Voona, and SM Sameer. Smart glove with gesture recognition ability for the hearing and speech impaired. In 2014 IEEE Global Humanitarian Technology ConferenceSouth Asia Satellite (GHTC-SAS), pages 105–110. IEEE, 2014.

[11] Yasmeen Raushan, Abhishek Shirpurkar, Vrushabh Mudholkar, Shamal Walke, Tejas Makde, and Pratik Wahane. Sign language detection for deaf and dumb using flex sensors. Int. Res. J. Eng. & Technol (IRJET), 4(3):1023–1025, 2017.