Indian sign language to voice using ESP32_Cam for hand tracking and MediaPipe for hand gesture detection

Dissertation submitted in fulfilment of the requirements for the Degree of

BACHALOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

Arjun Lande 12107702

Supervisor

Mr. Ajay Sharma

Lovely Professional University

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Lovely Professional University

Phagwara, Punjab (India)

Month: May, Year:2025

@ Copyright LOVELY PROFESSIONAL UNIVERSITY, Punjab (INDIA)

DECLARATION STATEMENT

I hereby declare that the project work entitled "**Indian sign language to voice using ESP32_Cam for hand tracking and MediaPipe for hand gesture detection**" is an authentic record of our own work carried out as requirements of Capstone Project for the award of B. Tech degree in Computer Science and Engineering from Lovely Professional University, Phagwara, under the guidance of **Mr. Ajay Sharma**, during Jan to May 2025. All the information furnished in this capstone project report is based on our own intensive work and is genuine.

Project Group Number: K21UW(G15)

Name of Student: Arjun Lande
Registration Number: 12107702

(Signature of Student)
Date: 01/05/25

CERTIFICATE

This is to certify that the declaration statement made by this group of students is correct to the best of my knowledge and belief. They have completed this Capstone Project under my guidance and supervision. The present work is the result of their original investigation, effort and study. No part of the work has ever been submitted for any other degree at any University. The Capstone Project is fit for the submission and partial fulfilment of the conditions for the award of B.Tech degree in Computer Science and Engineering from Lovely Professional University, Phagwara.

Signature of Supervisor

Mr. Ajay Sharma

Date: 01/05/25

Counter Signed by:

1) **Concerned HOD**

   HOD's Signature: _____

   HOD Name: _____

   Date: _____

2) **Neutral Examiner:**

   External Examiner

   Signature: _____

   Name: _____

   Affiliation: _____

   Date: _____

   Internal Examiner

   Signature: _____

   Name: _____

   Date: _____

ACKNOWLEDGMENT

I would like to express my sincere gratitude to the students of **Lovely Professional University** for their valuable support and willingness to contribute in our project by providing their hand gesture data. The gesture recognition model for sign language improved dramatically because of the hand gesture data provided by the students. The process of system training alongside adjustment would not have been effective without their involvement.

I am deeply thankful to our research supervisor, **Mr. Ajay Sharma**, for his continuous guidance, insightful suggestions, and encouragement throughout the project. His mentorship helped shape the direction of this work and ensured its successful and timely completion.

I would also like to extend my heartfelt thanks to **UpGrad Campus** for their support. Their approval, along with the essential resources made it possible to implement and test the project effectively. Their backing played a significant role in transforming the research idea into a working, impactful solution.

This project is the result of collaborative effort, and I am truly grateful to everyone who contributed to its success.

TABLE OF CONTENTS

CONTENTS	PAGE NO.
Cover Page	I
Declaration by the Scholar	II
Supervisor's Certificate	III
Acknowledgement	IV
Table of contents	V
List of Tables	VIII
List of Figures	IX
Abstract	XI
CHAPTER 1: INTRODUCTION	1
1.1 Background	1
1.2 Problem Statement	1
1.3 Objective	2
1.4 Significance of the Study	2
1.5 Machine Learning and Numeric Label Classification	3
1.6 MediaPipe	4
1.7 Classification Models (Random Forest)	5
CHAPTER 2: LITERATURE REVIEW	7
2.1 Evolution of Assistive Technologies in Gesture Recognition	7
2.2 Glove-Based Gesture Recognition Systems	7
2.3 Vision-Based Gesture Recognition Approaches	8
2.4 MediaPipe: A Modern Framework for Hand Gesture Detection	9
2.5 Studies on Hand Gesture Detection Using MediaPipe	9
2.6 Integration of ESP32-CAM in Vision-Based Systems	10
2.7 Challenges and Limitations in Existing Approaches	10
2.8 Comparative Analysis of Gesture Recognition Approaches	11

2.9 Conclusion and Future Directions 12

CHAPTER 3: Research Methodology 13

3.1 Data Acquisition 13

3.1.1 External Camera for Video Streaming and Gesture Recognition 15

3.1.2 ESP32-CAM Module for Real-Time Hand Tracking 15

3.2 Preprocessing 16

3.2.1 Frame Extraction 16

3.2.1 Colour Space Conversion 16

3.3 Hand Landmark Detection and Gesture Segmentation 16

3.3.1 Palm Detection and Bounding Box Generation 16

3.3.2 Landmark Estimation 17

3.3.3 Gesture Segmentation 17

3.4 Gesture Classification and Model Training 17

3.4.1 Feature Extraction 17

3.4.2 Model Training 19

3.5 Text-to-Speech (TTS) Conversion 19

3.5.1 Gesture-to-Text Mapping 19

3.5.2 Speech Synthesis 20

3.6 System Integration 20

3.6.1 Hardware-Software Synergy 21

3.6.2 Servo-Controlled Camera Alignment 21

3.6.3 Modular System Design 21

CHAPTER 4: Result 22

4.1 Performance Metrics 22

4.2 Quantitative Results 23

4.3 Classification Report and Confusion Matrix 23

4.4 ROC Curve Analysis 26

4.5 Feature Importance (Random Forest Classifier) 27

4.6 GUI Implementation 29

4.7 Discussion 30

CHAPTER 5: Design 31

5.1 System Design 31

5.2 Pseudo code 32

CHAPTER 6: Testing 35

CHAPTER 7: Conclusion 36

CHAPTER 8: PROJECT CODE 38

8.1 Programme for Data Collection for our Project: 38

8.2 Programme for renaming collected images and organising before dataset creation 41

8.3 Programme for creating and saving dataset in pickle format 42

8.4 Programme for train model on pickle dataset 45

8.5 C/C++ Programme for hardware to communicate with servo motor for hand tracking 46

8.6 Programme for Final TTS inference without hardware integration 47

8.7 Programme for Final TTS GUI Hardware integrated inference with hardware included 55

CHAPTER 9: REFERENCE 66

Research paper (Status) 70

Plagiarism (Status) 72

LIST OF TABLES

TABLE NO.	DESCRIPTION	PAGE NO.
Table 1	Comparative Analysis	11
Table 2	Showing the land marks representation of palm, hand, finger tips, palmer flexion creases	18
Table 3	Metric and Values	23
Table 4	The classification report hand gesture recognition	23
Table 5	Main Program	3
Table 6	LANDMARK EXTRACTION AND PREPROCESSING	33
Table 7	GESTURE CLASSIFICATION	33
Table 8	TEXT-TO-SPEECH CONVERSION	33
Table 9	SERVO CONTROL DECISION LOGIC	34
Table 10	ESP32 COMMAND SENDER	34
Table 11	Functional Testing	35

LIST OF FIGURES

FIGURE NO.	FIGURE DESCRIPTION	PAGE NO.
FIGURE 1.	Hardware assembly and model output	6
FIGURE 2.	Data collection 1	14
FIGURE 3.	Data collection 2	14
FIGURE 4.	ESP32-CAM MODULE	15
FIGURE 5.	Mediapipe Landmarks Representation	18
FIGURE 6.	System Architecture Design data flow diagram for the Hand Tracking for Sign Language	20
FIGURE 7.	Confusion Matrix showing the result of correlation between the predicted label class and actual label class	26
FIGURE 8.	Receivers Operator Curve showing the value for each class	27
FIGURE 9.	Feature importance analysis	28
FIGURE 10.	GUI Interface showing the predicted word along with the voice narration for sign language.	29
FIGURE 11.	Dataflow Diagram	31
FIGURE 12.	Data Collection programme part-1	38
FIGURE 13.	Data Collection programme part-2	39
FIGURE 14.	Data Collection programme part-3	40
FIGURE 15.	Renaming and organising collected images part-1	41
FIGURE 16.	Renaming and organising collected images part-2	41
FIGURE 17.	Creating and saving dataset in pickle format part - 1	42
FIGURE 18.	Creating and saving dataset in pickle format part – 2	43
FIGURE 19.	Creating and saving dataset in pickle format part – 3	44
FIGURE 20.	Creating and saving dataset in pickle format part - 4	44

FIGURE 21. Train model on pickle dataset part - 1 45

FIGURE 22. Train model on pickle dataset part - 2 45

FIGURE 30. C/C++ Programme for ESP32-CAM part – 1 46

FIGURE 31. C/C++ Programme for ESP32-CAM part – 2 46

FIGURE 32. Final TTS inference without hardware part – 1 47

FIGURE 33. Final TTS inference without hardware part – 2 48

FIGURE 34. Final TTS inference without hardware part – 3 49

FIGURE 35. Final TTS inference without hardware part – 4 50

FIGURE 36. Final TTS inference without hardware part – 5 51

FIGURE 37. Final TTS inference without hardware part – 6 52

FIGURE 38. Final TTS inference without hardware part – 7 53

FIGURE 39. Final TTS inference without hardware part - 8 54

FIGURE 40. Final TTS GUI Hardware integrated inference part – 1 55

FIGURE 41. Final TTS GUI Hardware integrated inference part – 2 56

FIGURE 42. Final TTS GUI Hardware integrated inference part – 3 57

FIGURE 43. Final TTS GUI Hardware integrated inference part – 4 58

FIGURE 44. Final TTS GUI Hardware integrated inference part – 5 59

FIGURE 45. Final TTS GUI Hardware integrated inference part - 6 60

FIGURE 46. Final TTS GUI Hardware integrated inference part – 7 61

FIGURE 47. Final TTS GUI Hardware integrated inference part – 8 62

FIGURE 48. Final TTS GUI Hardware integrated inference part – 9 63

FIGURE 49. Final TTS GUI Hardware integrated inference part - 10 64

FIGURE 50. Final TTS GUI Hardware integrated inference part - 11 65

ABSTRACT

Communication gaps experienced by people with speech and hearing disabilities continue to be a major issue in the modern world, particularly when communicating with people who does not know sign language to bridge this gap, this study introduces a real-time Indian Sign Language (ISL) to Voice translation system based on an ESP32-CAM module for capturing hand gestures and Media-Pipe for precise hand landmark detection and gesture recognition. The system identifies ISL alphabets and numbers using effective hand tracking and gesture segmentation, providing smooth conversion of sign language to text as well as synthesized speech output. Media-Pipe's hand tracking improves the precision of gesture recognition by capturing 21 critical hand landmarks, while a light machine learning model determines the gestures. The identified text is then converted to voice with the help of text-to-speech (TTS) synthesis, enabling clear communication for the speech and hearing impaired. Differing from conventional systems depending on heavy hardware or being confined to English speech, this project focuses on an embedded, cost-effective solution via ESP32-CAM, thereby being portable. In addition, the flexibility in the system will enable future enhancement for multilingual even regional languages such as Malayalam, to extend inclusivity. This paper describes the implementation, architecture and experimental performance of the system, showing its efficiency and accuracy in real-time ISL gesture recognition and voice generation. The system proposed here has great potential in filling the communication gaps, enabling independence and social inclusion of hearing and speech disabled individuals.

Keywords: Indian Sign Language (ISL), ESP32-CAM, Media-Pipe, Hand Gesture Recognition, Text-to-Speech (TTS), Machine Learning, Random Forest.

CHAPTER 1

INTRODUCTION

1.1 Background

As the world's most populated nation India faces an enormous task to handle disabilities particularly hearing loss that stands as the primary sensory deficiency within the human population. WHO data reveals that significant hearing impairments affect 63 million Indians which corresponds to 6.3% of the national population. Including children below 15 years of age total 4.5 million among the 63 million people who suffer from disability. Hearing loss together with associated ear diseases reduces people's sound perception skills which reduces their capacity to speak and connect with others in socially.

People communicate their thoughts and emotions along with their messages and ideas through both non-verbal and verbal methods. The main channel of communication for people who are mute or deaf are sign languages. Most of society faces difficulties understanding sign language. Most people show openness to learning spoken languages although very few demonstrate willingness to study sign languages. The challenge is intensified in India because people speak numerous sign languages which include Mumbai along with Delhi, Kolkata, Bangalore-Madras, Alipur, Indian Sign Language (ISL) and others. The numerous and region-specific sign languages create challenges for the common population when attempting to master and effectively utilize them.

1.2 Problem Statement

The majority of current sign language recognition systems operate for American Sign Language while Indian Sign Language (ISL) user needs remain rarely supported by available technologies. Devices with text or speech features through gestures tend to have multiple operational shortcomings by being inaccurate and expensive and without language or cultural localization for India. The communication barrier between mute and deaf people in India and the general population prevents these individuals from participating fully in education along with employment and social activities. Poor communication between mute and deaf communities leads to social isolation while creating opportunities for exclusion which requires the immediate development of an inclusive affordable solution suited for local Indian need

1.3 Objective

The primary purpose of this research project is to developing a real-time system which recognizes Indian Sign Language (ISL) used to reduce communication barrier between mute- deaf individuals and general users.

This system uses:

1. The ESP32-CAM module controls servo motor of 2 axis pan tilt servo assembly to accurately track hand movement.
2. the system applies Google's MediaPipe framework to analyses and predict ISL hand gestures.
3. Recognized gestures then translated into English sentences and then converted into speck by Google's TTS framework.
4. Users can read predicted sentences on screen for better communication.

The proposed system removes the requirement of wearable sensors and specialized hardware which is very expensive, hard to wear for long time and very uncomfortable.

1.4 Significance of the Study

This research holds a significant potential for improving available and Openness for deaf and mute peoples in India. By integrating machine learning and computer vision, the proposed system not only automates the recognition of ISL gestures but also closes the communication gap between thoughts who understand sign language and thoughts who are not familiar with sign language.

- Increased Available: It gives opportunity to deaf and mute individuals to express themselves freely in everyday interactions, both in personal and professional life.
- Language Openness: It gives base for future implementation of other regional sign languages offering a multilingual platform for everyone to reach.
- Cost-Effectiveness: This system uses low-cost and easy to obtain hardware like the ESP32-CAM module.
- Educational and Social Impact: It provide better learning chances to deaf and mute children helps them feel more accepted by reducing the need for personal sign language interpreters.

With further development and community adoption, this system can be increasing its support to other regional dialects and context-aware gesture interpretation, making it even more impactful in reducing the communication gap for deaf and mute peoples.

Machine Learning and Numeric Label Classification

1.5 Machine Learning and Numeric Label Classification

Through statistical models and algorithms Machine Learning operates as an AI subdivision to enable systems to learn patterns which allow them to decide through unprogrammed data extraction. ML systems refine their operational capabilities through the process of data training instead of executing pre-established commands.

During numeric label classification we supply machines with data classification instruction through numbered labels. The classification process requires teaching the system to interpret hand gestures as follows:

- 0 = "Hello"
- 1 = "Thank you"
- 2 = "Yes"
- 3 = "No"

The system converts each hand movement into numerical data which includes finger position information through (x, y coordinates). These are the features. The system uses the provided features to evaluate them then return an accurate prediction result.

Core Idea:

A machine learning model aims to approximate a function $f: X \rightarrow Y$, where:

- X: Input features (e.g., images, sensor data).
- Y: Output labels (numeric for regression, categorical for classification).

Types of ML:

1. **Supervised Learning**: Uses labelled data $(X, Y)\ to\ learn\ f$.

- *Formula*: For a linear model, $\hat{y} = w^t x + b$, where w is the weight vector and b is the bias.

2. **Unsupervised Learning**: Finds patterns in unlabelled data.
3. **Reinforcement Learning**: Learns via trial-and-error interactions with an environment.

Key Components:

- **Training**: Adjust model parameters $(e.g.,\ w, b)$ to minimize a loss function (e.g., MSE).
- **Generalization**: Ability to perform well on unseen data (test set).

Basic Workflow:

1. **Data Collection**: Input features (like hand landmarks).
2. **Labelling**: Assign each data point a number.
3. **Training**: Feed labelled data into an ML model.
4. **Prediction**: Use the trained model to classify new data.

Common Formula:

A common formula in classification is:

$$\hat{y} = f(x)$$

Where:

- x = input features (like hand landmarks),
- f = trained model,
- $\hat{y}$= predicted label (like "0" for "Hello").

1.6 MediaPipe

The free Google-developed tool MediaPipe enables users to detect hands together with faces as well as multiple other objects using webcam technology. The tool operates Computer Vision through a process which appears magical yet functions in real-time activities.

Hand gesture recognition through MediaPipe Hand Tracking framework detects twenty-one landmarks which represent points on every hand. Through fingertip and joint and wrist detection MediaPipe can identify gestures and various control devices.

How It Works:

1. **Camera Input** – MediaPipe takes the video feed.
2. **Detection Model** – It finds the hands in the frame.
3. **Landmark Model** – It gives 21 (x, y, z) coordinates for each hand.

1.7 Classification Models (Random Forest)

Random Forest functions is one of the most widely known and used machine learning classification algorithms. Random Forest takes its name from being multiple decision trees combined with random data selection in tree building.

To identify a specific fruit between apple, orange and banana utilize colour characteristics together with shape characteristics and weight measurements. The algorithm requests opinions from one hundred experts then implements the outcome based on which response wins the majority. Random Forest functions using the principle of utilizing many decision trees while performing data selection randomly throughout the process.

How It Works:

1. **Create many decision trees** using different parts of the data.
2. Each tree makes a prediction (e.g., "banana").
3. The final prediction is the majority vote from all trees.

Formula:

$$\hat{y} = mode(\mathrm{T}_1(X), \mathrm{T}_2(X), \ldots, \mathrm{T}_n(X))$$

Where:

- T_i is the i-th tree,
- x is the input feature,
- $\hat{y}$ is the predicted class.

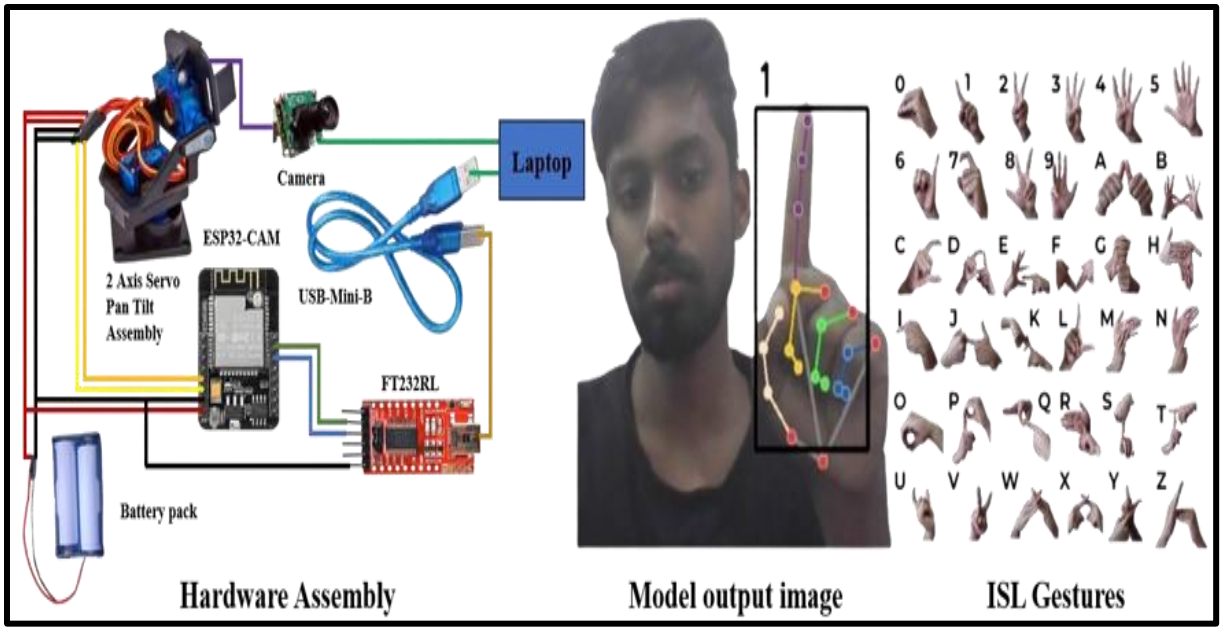

Fig. 1.1 Hardware assembly and model output

CHAPTER 2

LITERATURE REVIEW

2.1 Evolution of Assistive Technologies in Gesture Recognition

Since the last few decades researchers have devoted substantial effort to creating assistive technologies which benefit people with disabilities as well as those with speech and hearing difficulties. Early mechanical aids give the way for advanced AI-powered systems, and they have significantly improved since then. In the mid-20th century, assistive devices were primarily mechanical and analogue in nature, gives limited support to users. These early systems laid the groundwork for future innovations by highlighting the practical challenges faced by the peoples with disabilities and also opened new opportunity for digital alternatives.

The 1970s and 1980s marked the starting point of modernization in assistive technology with the development of basic screen readers, speech synthesis, and text-to-speech (TTS) systems. These tools weren't primarily designed for the visually impaired people but played a important role in creating the technological base for future developments. As machine learning and artificial intelligence (AI) gained significant growth in the 1990s and 2000s, these technologies enabled more responsive, real-time assistive tools capable of identifying facial expressions, audio cues, and, more recently hand gestures.

Gesture recognition, is an important component of this assistive technology, its earliest applications in controlled environments using handcrafted features. These features required precise and rigid movement sets that were difficult to reproduce naturally by users. As a result, systems based on handcrafted features faced major limitations in recognizing various and complex gestures commonly found in natural communication.

2.2 Glove-Based Gesture Recognition Systems

Glove-based gesture recognition systems are one of the earliest and most accurate system for capturing hand movement. One of the first designs where, the "Sayre Glove" from the 1970s, using flexible tubing with light sensors to detect finger bending and its positioning. Later innovations, such as the "Data Glove" and the "Power Glove," were equipped with inertial measurement units

(IMUs) and flex sensors to gives 3D tracking of finger and hand movements. These systems give high accuracy and were especially favoured in gaming, industrial applications, and sign language research.

However, despite their precision and accuracy, glove-based systems come with several disadvantages, including high cost, limited portability, and user discomfort. Users are required to wear the gloves continuously, making long-term usage difficult for users, especially for daily communication tasks. The maintenance and calibration of this glove-based system further add to the complexity of deploying such systems in large scale in real-world environments.

2.3 Vision-Based Gesture Recognition Approaches

Research efforts focus on vision-based approaches because they aim to eliminate glove-based system restrictions through static image analysis and video data gesture identification. The techniques eliminate contact requirements since they let users operate through a more convenient interface. Two main classifications exist within vision-based systems namely 3D model-based and appearance-based techniques.

3D model-based systems implement volumetric or skeletal models of hands both for detailed spatial data acquisition and motion data recording. The system delivers precise gesture evaluation through its technology although it needs advanced processors together with specialized recording devices such as depth cameras. The detection of gestures and their classification depends on appearance-based methods which analyse image or video content through templates or machine learning models. The systems need minimal hardware yet demand better lighting as well as being susceptible to background noises and partial blockage of camera view.

Modern deep learning techniques have transformed appearance-based methods through their developments. The current researches use a Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks to automatically detect gesture features. By understanding how gestures change over time, we can more accurately capture and interpret even the smallest of movements involved in sign language. Networked systems make systems more reliable and accurate when dealing with complicated gesture patterns as well as noisy conditions.

2.4 MediaPipe: A Modern Framework for Hand Gesture Detection

Google created MediaPipe as an open-source framework which enables real-time cross-platform machine learning pipelines. This powerful tool in gesture recognition emerged because it utilizes an efficient framework design with simple operation features along with capability to run on resource-efficient devices.

MediaPipe hand tracking pipeline consists of two principal components that include palm detection and hand landmark localization. The detection of palms in image frames occurs through a single-shot detection method within the first processing stage. The second stage employs regression-based models to precisely detect 21 hand landmarks when palms are identified. Hand landmarks provide precise analysis through their representation of hand key points starting from fingertips and moving to joints and ending at the wrist.

MediaPipe demonstrates great effectiveness in appearance-based vision systems through its resistance to lighting changes and its capability to run on low-power devices. The processing pipeline benefits from X and Y coordinate usage that achieves dependable accuracy during single-hand and dual-hand recognition.

2.5 Studies on Hand Gesture Detection Using MediaPipe

multiple studies have shown and strengthen the findings about the utilization of MediaPipe for detecting hand gestures. Researchers developed a real-time recognition framework for Indian Sign Language (ISL) which uses MediaPipe to obtain hand landmarks followed by gesture classification through a customized Convolutional Neural Network (CNN) model. A test evaluation with 30 Indian Sign Language (ISL) signs generated more than 95% precision for the proposed framework which demonstrated its application potential in assistive communication.

A different team applied MediaPipe to create a sign language tool which translated movements into different native languages across India. A Long short-term memory (LSTM) network was utilized by researchers to track Time pattern in gesture sequences which increased the recognition accuracy when dealing with dynamic signs. The research showed how MediaPipe operates seamlessly with deep learning systems while performing well with motion-based and movement-based gestures.

2.6 Integration of ESP32-CAM in Vision-Based Systems

The integration of ESP32-CAM along with its built-in camera module into gesture recognition systems allows researchers to create inexpensive portable solutions. The ESP32-CAM works poorly for direct AI processing but functions with a host computer to execute MediaPipe models.

An implementation based on the ESP32-CAM performed continuous hand detection as its primary function. The system operated in patrol mode by moving its camera across the environment through automated scanning until another hand appeared. The ESP32-CAM enabled hand detection after which it focused on the hand until sending video frames to the host PC for gesture analysis. The combination of host computer and ESP32-CAM delivery real-time tracking capabilities without excessive energy consumption.

2.7 Challenges and Limitations in Existing Approaches

Recent technological progress has not solved all problems with building reliable gesture-recognition technology because the following obstacles persist:

- Many vision-based systems encounter difficulties in detecting gestures accurately when the light conditions are either insufficient or too bright.
- Background noise together with hand occlusion as well as complex backgrounds create situations that usually confuse detection algorithms.
- Low-cost devices seem limited by their ESP32-CAM hardware restraint because they possess restricted processing capacity and memory space. This barrier hinders the direct deployment of complex models on the system.
- It proves difficult to create universal models because users exhibit diverse hand shapes along with various skin colours and gesture performance styles.
- Standardized datasets remain scarce because there is a lack of sufficient labelled data collections specifically for Indian Sign Language.

2.8 Comparative Analysis of Gesture Recognition Approaches

Table 2.0.1 comparative analysis

Feature	***MediaPipe + ESP32-CAM***	***Leap Motion (Infrared-Based)***	***Glove-Based Systems (Sensor-Based)***
Hardware Requirement	ESP32-CAM for Hand tracking + PC for processing	Leap Motion IR sensor module	Wearable gloves with flex & IMU sensors
Processing Power	Uses ESP32 for Hand tracking, PC for AI	High (specialized IR tracking)	External microcontroller or PC
Accuracy	High	Very High	Very High
3D Motion Capture	No	Yes	Yes
Lighting Dependency	Affected	Works in all conditions	Works in all conditions
Ease of Use	Moderate	Easy	Hard
Cost	Low (₹1200-₹1600 (plus PC cost))	Moderate (₹8000-₹10,000)	High (₹10,000-₹50,000+)
Portability	High	Moderate	Low
Best Use Case	Low-cost real-time gesture recognition	VR/AR, touchless interaction, gaming	Industrial applications, medical rehabilitation

2.9 Conclusion and Future Directions

Gesture recognition technologies undergo continuous development using mechanical gloves as a starting point which led to current AI-based vision systems. MediaPipe together with ESP32-CAM along with other affordable modules makes gesture recognition technology available and efficient for real-time sign language interpretation applications.

Until now the field encounters hurdles because of environmental sensitivity issues along with hardware restrictions and insufficient data storage capacity. The development of ISL datasets requires sustained effort because researchers should enhance real-time capabilities on embedded hardware while adding support for India's diverse languages. The solution of these system issues will make gesture recognition systems into effective tools for promoting inclusive interaction across diverse communication environments.

CHAPTER 3

RESEARCH METHODOLOGY

A thorough description appears in this chapter about how the real-time hand gesture recognition system works together with servo-based camera tracking and text-to-speech (TTS) feedback. This system merging computer vision tools with machine learning models and embedded hardware together with speech conversion allows users to communicate through hand gestures. The basic system components contain an external camera for MediaPipe gesture detection and an ESP32-CAM module which controls servos while performing hand tracking. The system also implements a voice output engine from gestures. The methodology is structured into six primary components: (3.1) Data Acquisition, (3.2) Preprocessing, (3.3) Hand Landmark Detection and Gesture Segmentation, (3.4) Gesture Classification and Model Training, (3.5) Text-to-Speech (TTS) Conversion, and (3.6) System Integration.

3.1 Data Acquisition

The first stage of our project involves data acquisition from separate yet connected components which include an external USB camera. The external camera features on this setup benefit us in gathering heigh quality gesture images for mediapipe frameworks to plot hand landmarks on this image then these 21 landmarks give X and Y coordinates for single hand get stored in pickle file to train our model.

Fig.3.1 Data collection 1

Fig.3.2 Data collection 2

3.1.1 External Camera for Video Streaming and Gesture Recognition

An external USB camera operates as the primary device to perform gesture detection. The video streaming functionality allows the processor to observe real-time gestures due to the camera system. A camera setup provides unblocked visual access to user's hands while transmitting the video output to the screen in real-time. The processing system analyses frames for ISL gestures of

both static and dynamic type. The external camera achieves precise landmark extraction and gesture segmentation because of its high resolution along with fast frame rate.

3.1.2 ESP32-CAM Module for Real-Time Hand Tracking

ESP32-CAM module serves the purpose of real-time hand tracking to steer an external camera that is mounted on servo motors. The module takes images with a resolution level for hand position detection without image detail. The camera's servo motors receive directional information from the ESP32-CAM through its X and Y coordinates determination which enables the motor to track hand positions. The purpose-built tracking function lessens main recognition system operational workload while enabling faster system responses.

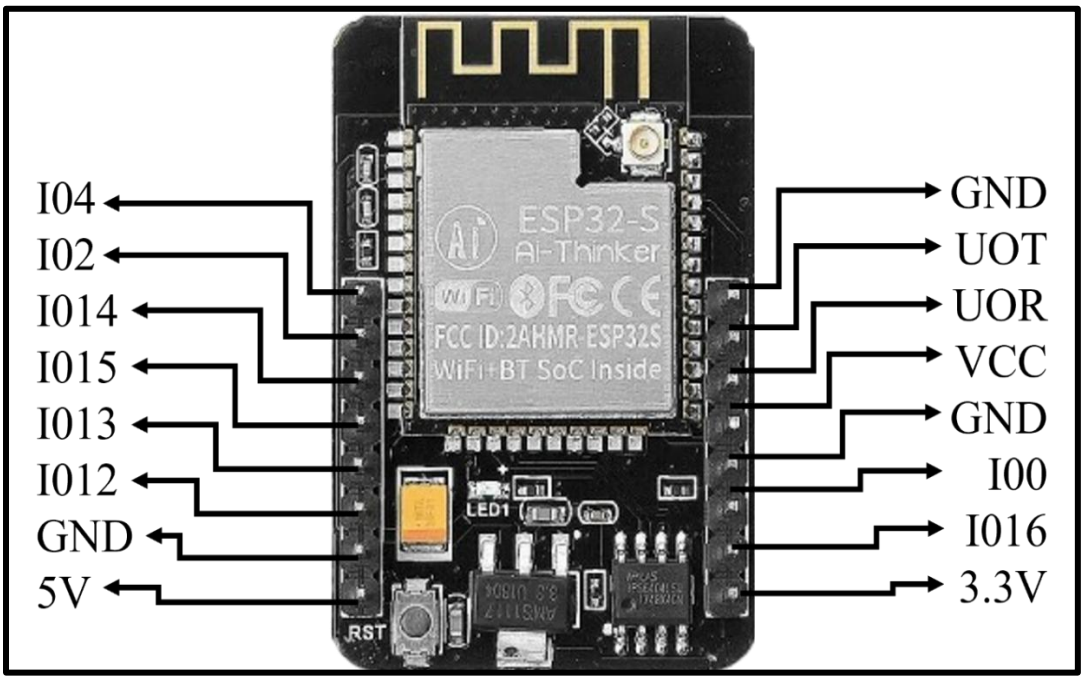

Fig.3.3 ESP32-CAM MODULE

3.2 Preprocessing

Video data preparation occurs during the preprocessing phase for both gesture recognition and servo control operation. During this processing phase of data, the framework requires information that meets MediaPipe specifications for precise hand landmark recognition.

3.2.1 Frame Extraction

After decomposing outdoor camera video through OpenCV image processing libraries the system receives individual image frames as inputs. The MediaPipe hand detection model receives single static images from each frame. The extraction process of real-time frames occurs every 24 to 30 times per second to maintain continuous temporal flow.

3.2.2 Colour Space Conversion

Each frame is transform from BGR (Blue-Green-Red) to RGB (Red-Green-Blue) colour space, which is the required format for mediapipe processing. The operation implements normalization alongside resizing steps to maintain frame dimension consistency for stable landmark detection.

3.3 Hand Landmark Detection and Gesture Segmentation

The fundamental element of the gesture recognition system depends on the MediaPipe Hands recognition framework that Google developed. MediaPipe Hands recognition framework provides robust real-time gesture detection together with tracking functions that makes it suitable for applications needing accurate and fast landmark retrieval capabilities.

3.3.1 Palm Detection and Bounding Box Generation

MediaPipe Hands begins by detecting the presence of a palm in each RGB frame. A random forest model identifies and generates a bounding box around detected palms to localize hands within the frame.

3.3.2 Landmark Estimation

Following palm detection, MediaPipe estimates 21 hand landmarks per hand, yielding 84 X and Y coordinates for two-handed gestures. These landmarks include joints, fingertips, and the wrist, forming a comprehensive spatial representation of hand position.

3.3.3 Gesture Segmentation

The system analysis temporal hand movements of landmarks to detect posture and motion modifications. The system creates gesture boundaries through significant variations which divides ongoing movement into separate gestures. The method of segmentation serves a critical purpose to separate distinct gestures from each other including fluid sign movements.

3.4 Gesture Classification and Model Training

The extracted landmarks function as a basis for detecting gestures inside predefined categories of ISL. The model training as well as model validation and live predictions utilize machine learning technology within this stage.

3.4.1 Feature Extraction

A feature vector results from flattening the 21 landmark coordinates. The model can benefit from extra derived characteristics such as distance measurements between landmarks and joint movement estimations. The algorithm ingests these characteristics as its input elements.

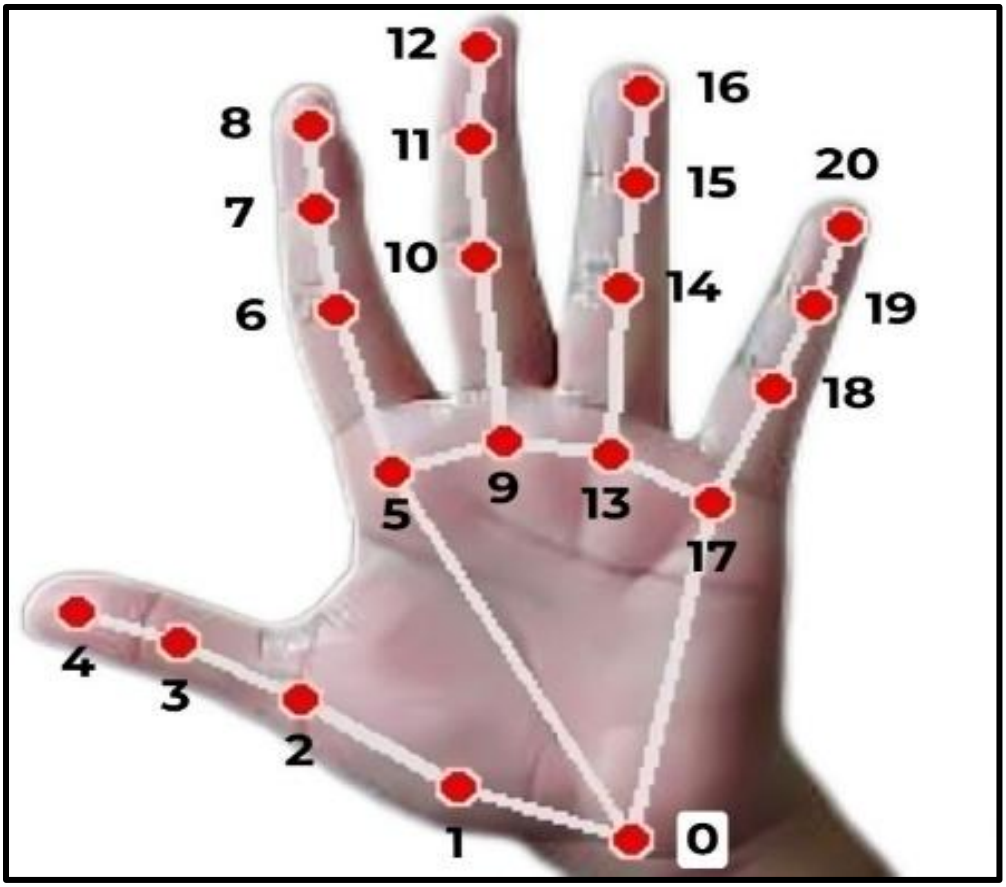

Fig. 3.4 Mediapipe Landmarks Representation

Table 3.0.1 Showing the land mark representation of Palm, Hand, Finger Tips, Palmer Flexion Creases

0	WRIST	**11**	MIDDLE_FINGER_DIP
1	THUMB_CMC	**12**	MIDDLE_FINGER_TIP
2	THUMB_MCP	**13**	RING_FINGER_MCP
3	THUMB_IP	**14**	RING_FINGER_PIP
4	THUMB_TIP	**15**	RING_FINGER_DIP
5	INDEX_FINGER_MCP	**16**	RING_FINGER_TIP

0	WRIST	**11**	MIDDLE_FINGER_DIP
6	INDEX_FINGER_PIP	**17**	PINKY_MCP
7	INDEX_FINGER_DIP	**18**	PINKY_PIP
8	INDEX_FINGER_TIP	**19**	PINKY_DIP
9	MIDDLE_FINGER_MCP	**20**	PINKY_TIP
10	MIDDLE_FINGER_PIP		

3.4.2 Model Training

A machine learning model comprising of Random Forest or Support Vector Machine (SVM) uses a set of ISL gestures that have been properly labelled to establish itself. The generalization of models improves through data augmentation methods that involve changing images by rotation or flipping them and adjusting their size. The dataset completes a k-fold cross-validation process to both optimize coefficients and measure how well the model functions. The selection of models depends on accuracy measurements combined with precision levels and recall rates as well as F1-score results.

3.5 Text-to-Speech (TTS) Conversion

Identified gestures transform into textual meanings that get assembled into verbal expressions or sentences. The engine or equivalent TTS module transforms the assembled textual meanings into speech with assistance from Google's Text-to-Speech (gTTS) engine.

3.5.1 Gesture-to-Text Mapping

The gestures which the system understands get linked to particular words or letters. The sentence construction system comprises a buffer mechanism which gathers gesture sequences then detects word limits through movement duration combined with geographic margins.

3.5.2 Speech Synthesis

The predicted text string goes to a TTS called Google Text-to-Speech (gTTS) before it plays the audio through speakers. After converting text to audio, the TTS sends the resulting audio through the system speakers. The system provides verbal communication to users who are deaf and mute cannot communicate through speech.

3.6 System Integration

The system ends with integration of video detection and gesture reorganization mechanisms along with hand detection algorithms and servo controller operations and audio outcome functionality. The unified system works harmoniously because all subsystems connect with each other without delays or conflicting operations.

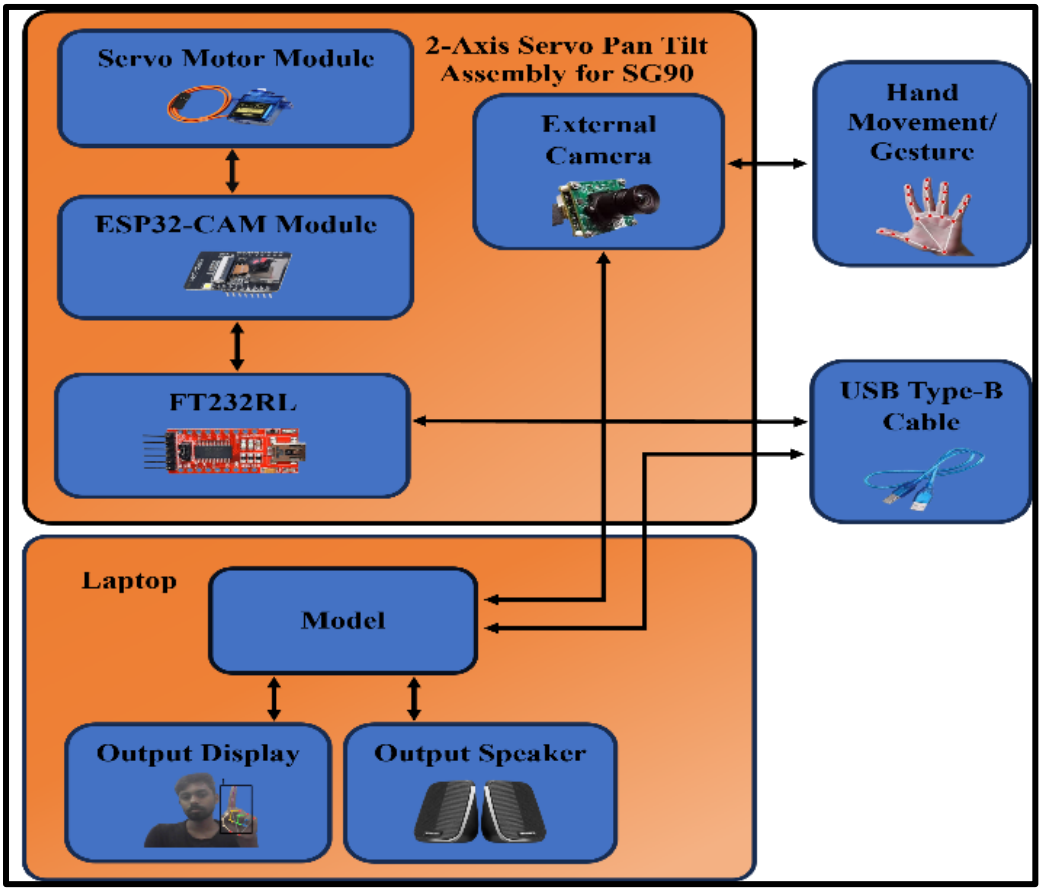

Fig. 3.5 System Architecture Design data flow diagram for the Hand Tracking for Sign Language

3.6.1 Hardware-Software Synergy

The system divides gesture recognition functions between various devices from tracking ones. The system utilizes external camera implementation to detect landmarks using CPU of the system yet relies on ESP32-CAM to do tracking and servo control tasks independently. The divided system tasks increase both system speed and efficiency.

3.6.2 Servo-Controlled Camera Alignment

Actions from the ESP32-CAM lead the motors to receive commands about hand coordinate positions X and Y. The motors achieve exterior camera orientation through their commands to keep the user's hand in view. Through this tracking mechanism the gestures remain visible to the camera leading to better recognition accuracy.

3.6.3 Modular System Design

Each module is independently upgradable. Because gesture models operate independently from the hardware there is no need to alter physical components while training the system. The tracking system maintains flexibility for adapting to various robotic applications. Future modification and scaling of the system becomes achievable because of its modular design.

CHAPTER 4

RESULT

This section provides evaluations about the real-time speech generation system from ISL hand gestures which operates through computer vision and machine learning algorithms. Different performance metrics served as the foundation for extensive testing of the hand gesture recognition model while the confusion matrix alongside ROC curves together with feature importance analysis contributed to visual performance evaluation. Testing on the graphical user interface's usability and effectiveness took place alongside the rest of the evaluation process.

4.1 Performance Metrics

The Indian Sign Language (ISL) gesture recognition system was tested using standard classification metrics including precision, accuracy, F1-score and recall. These metrics provide a complete understanding of the model's ability to correctly recognize and classify ISL gestures.

The following formulas were used:

- **Accuracy**: Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
- **Precision**: Precision = $\frac{TP}{TP+FP}$
- **Recall (Sensitivity)**: Recall = $\frac{TP}{TP+FN}$
- **F1 Score**: F1-Score = $2 \times \frac{Precision \times Recal}{Precision+Recal}$

Where TN = True Negative, TP = True Positive, FN = False Negative, FP = False Positive.

4.2 Quantitative Results

Table 4.0.1 Metric and Value

Metric	***Value***	***Metric***	***Value***
Accuracy	95.67%	F1-Score	0.97
Precision	0.98	Min_Samples_Leaf	30
Recall	0.96	Random State	42

These metrics confirm that the system is capable of recognizing ISL gestures with high accuracy and converting them into text.

4.3 Classification Report and Confusion Matrix

The classification report for various ISL gestures shows the individual class-wise recall, precision, F1-score, and support:

Table 4.2 The classification Report of the Hand Gesture Recognition, Showing the Class Recall, Precision, F1-Score, Support

Class	***Precision***	***Recall***	***F1-Score***	***Support***
0	0.39	1.00	0.56	480
1	1.00	0.95	0.97	480
2	0.97	0.95	0.96	479
3	1.00	0.98	0.99	479
4	1.00	0.95	0.98	480
5	1.00	0.94	0.97	480
6	0.99	0.98	0.99	480
7	1.00	0.95	0.98	480

8	1.00	0.98	0.99	480
9	1.00	0.97	0.98	480
10	1.00	0.98	0.99	457
11	1.00	0.98	0.99	474
12	1.00	0.99	1.00	480
13	1.00	0.93	0.96	480
14	1.00	0.96	0.98	450
15	1.00	1.00	1.00	450
16	1.00	0.96	0.98	480
17	1.00	0.98	0.99	467
18	1.00	0.91	0.95	434
19	1.00	0.96	0.98	377
20	1.00	1.00	1.00	480
21	1.00	0.97	0.98	462
22	1.00	0.91	0.95	476
23	1.00	0.98	0.99	476
24	0.98	0.92	0.95	406
25	1.00	0.84	0.91	455
26	1.00	0.95	0.97	470
27	1.00	0.98	0.99	480
28	1.00	0.95	0.98	377
29	1.00	0.91	0.95	398
30	1.00	0.99	0.99	433
31	1.00	1.00	1.00	448
32	1.00	0.96	0.98	479
33	1.00	0.96	0.98	465

34	1.00	0.90	0.94	474
35	1.00	0.97	0.98	438
36	1.00	0.96	0.98	480
37	1.00	0.88	0.94	402
38	1.00	0.99	0.99	479

- **Macro Average:** Precision = 0.98, Recall = 0.96, F1-score = 0.97
- **Weighted Average:** Precision = 0.98, Recall = 0.96, F1-score = 0.97
- **Overall Accuracy:** 96% (17,925 total samples)

A confusion matrix (Figure 4.1) further illustrates the correlation between the predicted and actual gesture classes. Each diagonal cell represents correctly predicted gestures, while off-diagonal values indicate misclassifications. The matrix helps identify specific gestures where errors may occur.

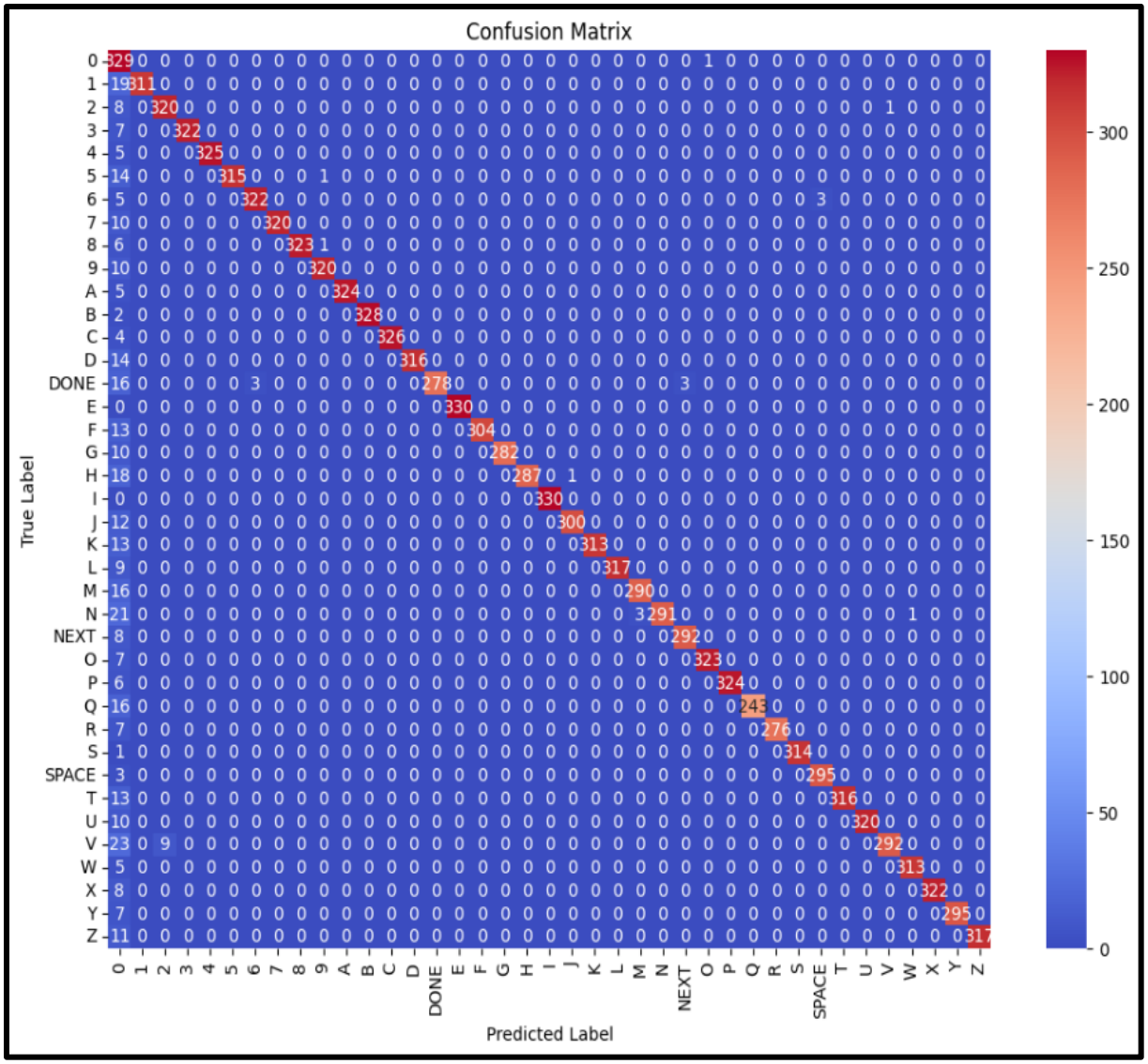

Fig. 4.1 Confusion Matrix showing the result of correlation between the predicted label class and actual label class

4.4 ROC Curve Analysis

The Receiver Operating Characteristic (ROC) curve was created for each class to visualize the model's capability to differentiate between multiple classes. The Area Under the Curve (AUC) is close to 1.0 for all classes, which shows that the classifier's ability to make accurate predictions. A high AUC represents low false positive rates and high true positive rates.

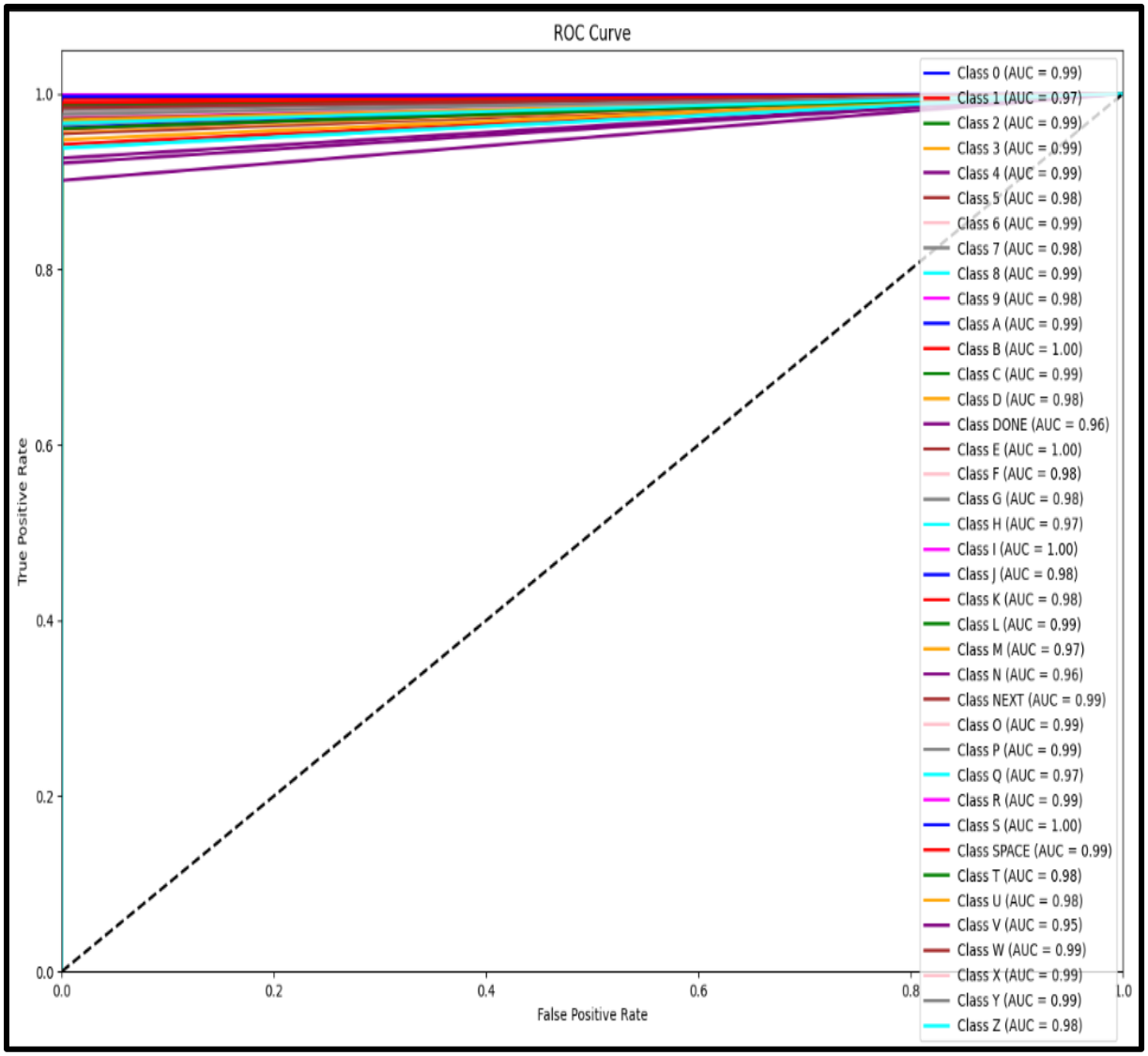

Fig.4.2 Receivers Operator Curve showing the value for each class

4.5 Feature Importance (Random Forest Classifier)

Using a Random Forest classifier also enabled the evaluation of which hand landmarks contributed most significantly to gesture recognition. The feature importance diagram (Figure4.3) shows that fingertips and joint regions hold the highest weight in determining the gesture type. This insight allows the system to be further optimized by focusing on these high-weight landmarks.

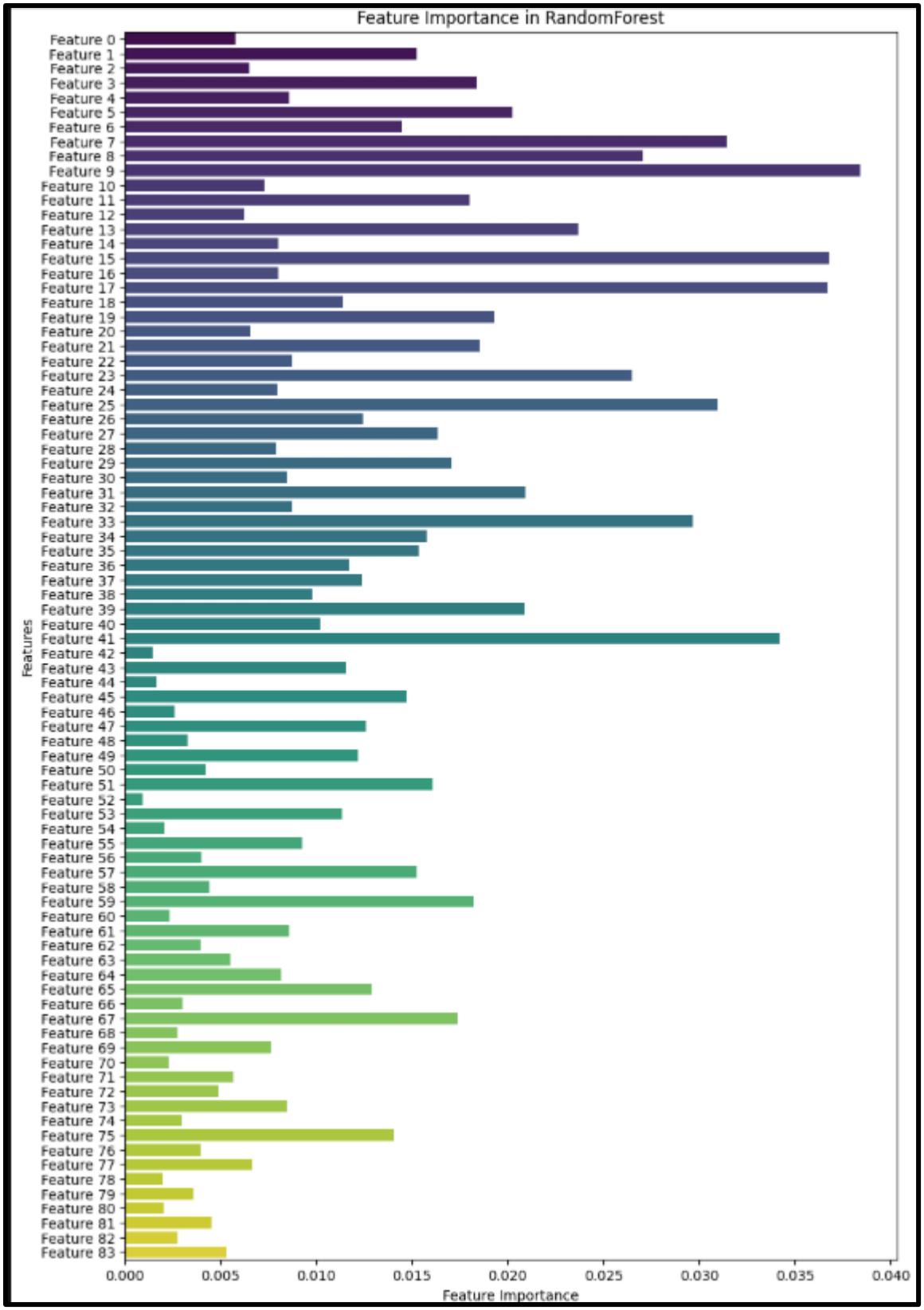

Fig.4.3 Feature importance analysis

4.6 GUI Implementation

The user interface was designed with accessibility in mind, ensuring ease of use for individuals with hearing or speech disabilities. The GUI includes:

- Real-time hand gesture video feed
- Predicted gesture label display
- Text transcription window
- Audio output using Google Text-to-Speech (gTTS)

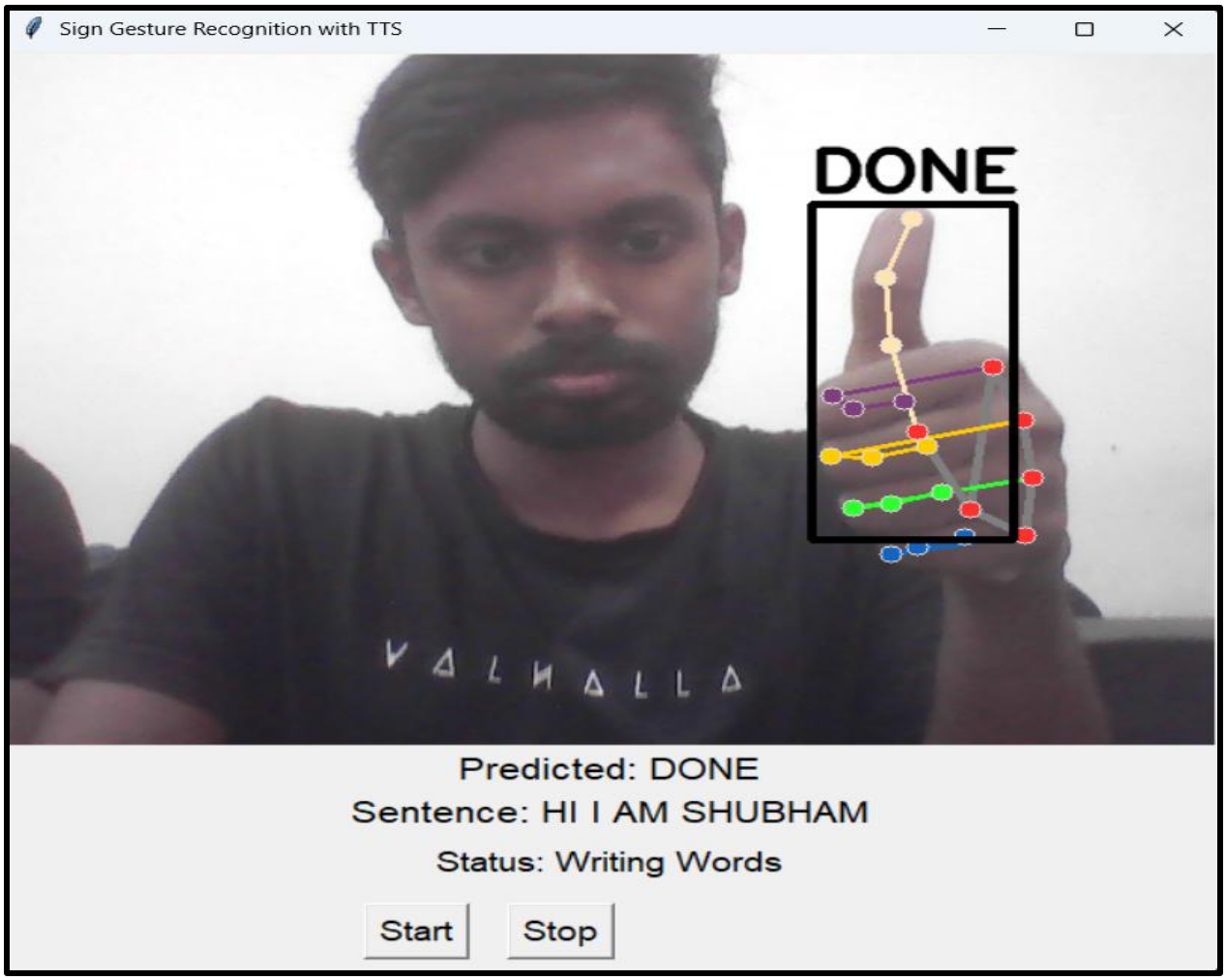

Fig. 4.4 GUI Interface showing the predicted word along with the voice narration for sign language.

Figure 4.4 demonstrates the GUI with a recognized gesture, its corresponding text, and the speech playback. This setup enables seamless interaction and communication

4.7 Discussion

The results affirm the efficacy of the modal in translating ISL gestures into text and speech with relatively high accuracy. The integration of MediaPipe's robust hand tracking framework with the Random Forest classifier ensures:

- **High Real-time Performance:** Quick response and accurate feedback
- **Cost Efficiency:** Utilization of ESP32-CAM for servo tracking reduces hardware costs
- **Modularity and Scalability:** The system is adaptable and can be expanded with additional gesture classes or support for other languages

In comparison to other methods, our approach offers a lightweight, scalable, and accurate solution for gesture-based communication systems tailored for the Indian context.

CHAPTER 5

DESIGN

5.1.System Design

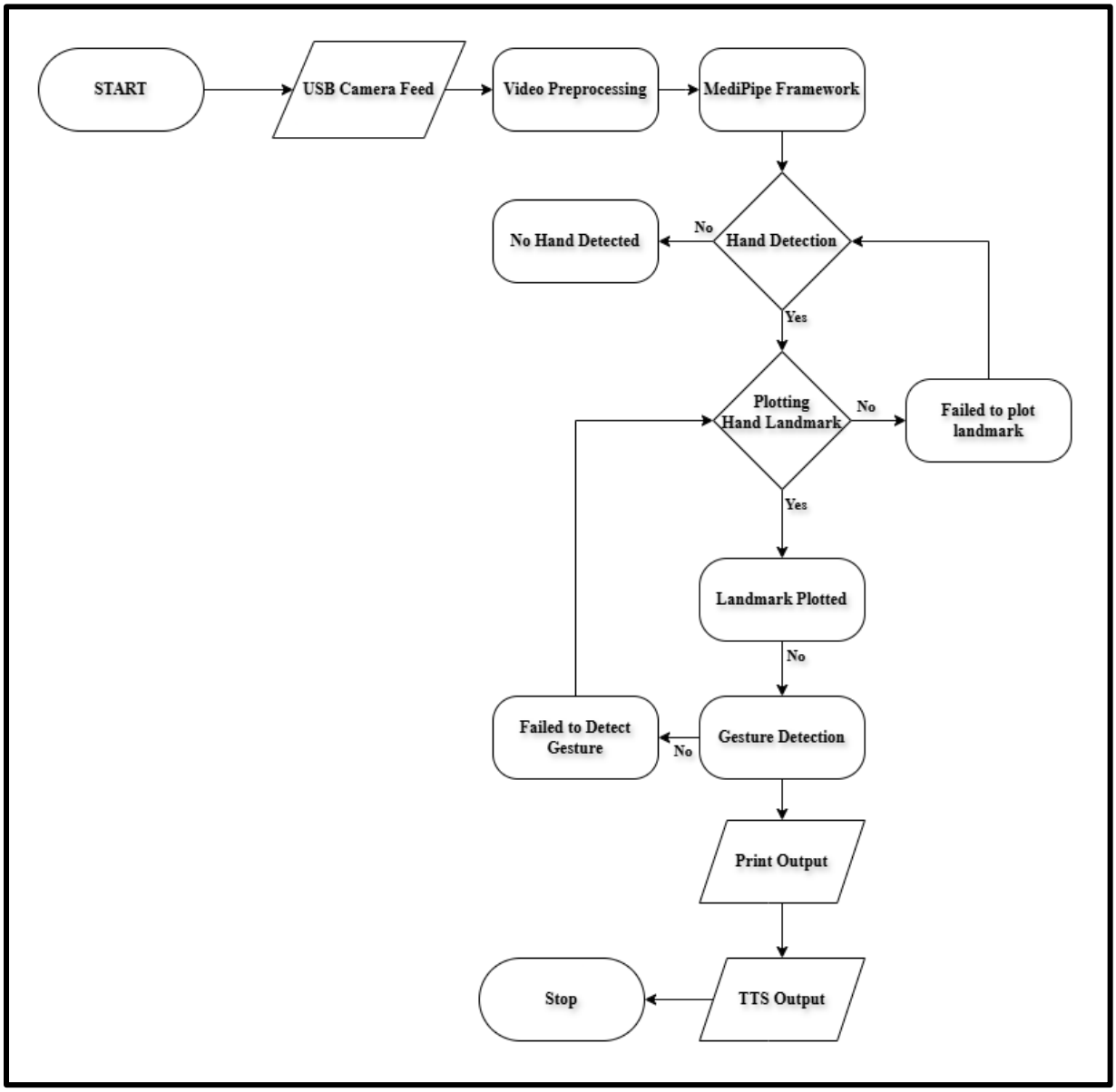

Fig.6.1 Dataflow Diagram

5.2.Pseudo code

Main Program Execution

BEGIN
  Initialize USB camera
  Load trained gesture classification model
  Initialize MediaPipe Hand module
  Connect to ESP32 via serial port
  Start GUI window
 	
  WHILE GUI is running:
  Read frame from USB camera
  Detect hands using MediaPipe
  IF hands are detected THEN
  Extract landmarks
  Preprocess landmark data
  Predict gesture using classifier
  Display gesture on GUI
  Convert gesture to audio using TTS
  Determine hand position for servo control
  Send command to ESP32 based on hand position
  ELSE
  Display "No Hand Detected"
  Send "IDLE" to ESP32
  ENDIF
  ENDWHILE
END

Table.5.1. Main Program

LANDMARK EXTRACTION AND PREPROCESSING

FUNCTION EXTRACT_LANDMARKS(HAND_LANDMARKS):
  FOR EACH HAND IN HAND_LANDMARKS:
  LANDMARK_VECTOR = []
  FOR EACH LANDMARK IN HAND:
  NORMALIZE X, Y, Z COORDINATES
  APPEND TO LANDMARK_VECTOR
  ENDFOR
  RETURN LANDMARK_VECTOR
END FUNCTION

Table.5.2. LANDMARK EXTRACTION AND PREPROCESSING

GESTURE CLASSIFICATION

FUNCTION CLASSIFY_GESTURE(LANDMARK_VECTOR):
  IF MODEL IS TRAINED:
  GESTURE = MODEL.PREDICT(LANDMARK_VECTOR)
  RETURN GESTURE
  ELSE
  RETURN "UNKNOWN"
  ENDIF
END FUNCTION

Table.5.3. GESTURE CLASSIFICATION

TEXT-TO-SPEECH CONVERSION

FUNCTION SPEAK_GESTURE(GESTURE_TEXT):
  TTS = CREATE GTTS OBJECT WITH GESTURE_TEXT
  SAVE TTS TO AUDIO FILE
  PLAY AUDIO FILE
END FUNCTION

Table.5.4. TEXT-TO-SPEECH CONVERSION

SERVO CONTROL DECISION LOGIC

FUNCTION DETERMINE_HAND_POSITION(HAND_X, FRAME_CENTER_X, BUFFER_ZONE):
  IF HAND_X < (FRAME_CENTER_X - BUFFER_ZONE):
  RETURN "TRACK:LEFT"
  ELSE IF HAND_X > (FRAME_CENTER_X + BUFFER_ZONE):
  RETURN "TRACK:RIGHT"
  ELSE
  RETURN "CENTER"
  ENDIF
END FUNCTION

Table.5.5. SERVO CONTROL DECISION LOGIC

ESP32 COMMAND SENDER

FUNCTION SEND_TO_ESP32(COMMAND):
  IF SERIAL CONNECTION IS ACTIVE:
  WRITE COMMAND OVER SERIAL
  ELSE:
  DISPLAY "ESP32 NOT CONNECTED"
  ENDIF
END FUNCTION

Table.5.6. ESP32 COMMAND SENDER

CHAPTER 6

TESTING

Module	Test Case	Expected Output	Result
Gesture Detection	Detect single or both hands using MediaPipe	Accurate landmark extraction	Passed
Gesture Classification	Classify static ISL gestures from hand landmarks	Correct label prediction	Passed
GUI Display	Show live feed and recognized gesture label	Real-time frame update and label visibility	Passed
TTS Conversion	Convert predicted label to speech	Clear audio output in English	Passed
ESP32 Servo Tracking	Move servo based on hand position	Smooth servo motion toward hand centre	Passed
Serial Communication	Send/receive messages between PC and ESP32	Correct parsing and response	Passed

Table.6.1. Functional Testing

CHAPTER 7

CONCLUSION

The hardware application of our project delivers 95.67% accuracy in understanding Indian Sign Language through machine learning model. This system functions primarily to make communication easier between deaf and mute people and other speakers without human interpretation assistance. The process of detecting hand gestures and speech translation operates through artificial intelligence (AI) together with machine learning.

The special feature of our model enables users to operate it using hardware or completely without hardware components. Our model works with either laptop or phone camera operations or it enables additional tracking capabilities through the ESP32-CAM module.

Our system has three main features:

1. **Sign Language Recognition**
   Google developed MediaPipe as a tool to perform this feature. MediaPipe functions as a detector which tracks finger movements through joint position analysis. Different sign language gestures get identified by processing the detected hand landmarks through this system. The classification system identifies different gestures through machine learning while it compares them to their corresponding meanings.

2. **Text-to-Speech (TTS)**
   The system uses gTTS Text-to-Speech from Google to transform recognized text into spoken words after conversion. This process allows smooth communication between mute-deaf individuals and people with hearing capabilities.

3. **Hand Tracking with ESP32-CAM**
   Our system implements the ESP32-CAM as an affordable small camera tool. The ESP32-CAM operates as a minimal camera system which performs real-time hand movement monitoring. Together with MediaPipe the system becomes interactive since the camera tracks hand motions through the combination of the features. The system becomes more

usable with increased user freedom of movement because hand position does not need to stay at camera's centre.

Our project emphasis lay heavily on developing a hand gesture database. The use of MediaPipe allowed us to find precise finger and hand positions ensuring clear image characteristics for better dataset creation. The precise hand landmark information that MediaPipe provided enabled us to create a machine learning model that was more precise. Hand landmark data provided by MediaPipe offers higher precision than existing public database hand information.

Industrial development of the project posed several issues for our team. The project faced significant difficulty when processing live video footage due to its heavy computational demands. A slowness issue occurs when the system fails to identify gestures without delay. The system suffers from slow processing times mainly because ESP32-CAM operates alongside gesture recognition features. The problems can be addressed by employing future cloud computing platforms which will manage extensive processing requirements.

Looking ahead, we see great potential for our system. In the future, we want to:

- The platform should enable users to operate through different Indian languages to facilitate wider national adoption.
- Develop a single standalone software application to enable easy use by all device owners for downloading and employing.
- Our focus includes modifying the user interface (UI) to establish a system that functions efficiently with simplicity as well as user-friendliness and clearness.

The developed model serves as a significant advancement toward simplifying communication methods available to deaf and mute communities. The integration of AI and gesture recognition together with speech technology enables our system to connect users of sign language with users who lack this capability. This system operates at schools as well as hospitals and public spaces and domestic environments to simplify all daily conversations among all users.

CHAPTER 8

SOURCE CODE

1. Programme for Data Collection for our Project:

```python
import os
import cv2
import mediapipe as mp
import tensorflow as tf
import threading
tf.constant(0)

cap = cv2.VideoCapture(1)

# Flag to stop live feed
global stop_live_feed
stop_live_feed = False


def start_live_feed():  1 usage  shubhamsony1
    while not stop_live_feed:
        ret, frame = cap.read()
        if not ret:
            continue
        frame = cv2.resize(frame,  dsize: (500, 380))  # Resize frame
        cv2.imshow( winname: 'Live Feed', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break


live_feed_thread = threading.Thread(target=start_live_feed, daemon=True)
live_feed_thread.start()

reg_no = input("\nEnter Registration Number: ")

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
```

Fig. 7.1. Data Collection programme part-1

```python
hands = mp_hands.Hands(static_image_mode=False, min_detection_confidence=0.3)

BASE_DIR = r'C:\Users\ASUS\Documents\sem 8 new\Indian_sign_lang_recognition_model\DATA_COL\imgs_without_landmarks'
BASE_DIR_L = r'C:\Users\ASUS\Documents\sem 8 new\Indian_sign_lang_recognition_model\DATA_COL'
DATA_DIR = os.path.join(BASE_DIR, reg_no)
LANDMARKS_DIR = os.path.join(BASE_DIR_L, 'imgs_with_landmark', reg_no)

if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)
if not os.path.exists(LANDMARKS_DIR):
    os.makedirs(LANDMARKS_DIR)

dataset_size = 100

while True:

    gesture_name = input("\nEnter gesture name (or type 'quit' to exit): ")
    if gesture_name.lower() == 'quit':
        break

    class_dir = os.path.join(DATA_DIR, gesture_name)
    landmark_class_dir = os.path.join(LANDMARKS_DIR, gesture_name)

    if not os.path.exists(class_dir):
        os.makedirs(class_dir)
    if not os.path.exists(landmark_class_dir):
        os.makedirs(landmark_class_dir)

    print(f'Collecting data for gesture: {gesture_name}')

    counter = 0
    while counter < dataset_size:
```

Fig. 7.2. Data Collection programme part-2

```python
        ret, frame = cap.read()
        if not ret:
            continue

        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = hands.process(frame_rgb)
        frame_with_landmarks = frame.copy()

        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
                mp_drawing.draw_landmarks(frame_with_landmarks, hand_landmarks, mp_hands.HAND_CONNECTIONS)

        frame = cv2.resize(frame, dsize: (500, 380))  # Resize original frame
        frame_with_landmarks = cv2.resize(frame_with_landmarks, dsize: (500, 380))  # Resize landmarks frame

        cv2.imshow( winname: 'Original Frame', frame)
        cv2.imshow( winname: 'Frame with Landmarks', frame_with_landmarks)

        img_path = os.path.join(class_dir, f'{counter}.jpg')
        landmark_img_path = os.path.join(landmark_class_dir, f'{counter}.jpg')
        cv2.imwrite(img_path, frame)
        cv2.imwrite(landmark_img_path, frame_with_landmarks)
        counter += 1
        cv2.waitKey(25)

    print("Image collection complete!")

stop_live_feed = True  # Stop live feed thread
cap.release()
cv2.destroyAllWindows()
exit()
```

Fig.7.3. Data Collection programme part-3

2. Programme for renaming collected images and organising before dataset creation

```python
import os
import shutil

# Define paths
source_base = r"C:\Users\ASUS\Documents\sem 8 new\Indian_sign_lang_recognition_model\DATA_COL\imgs_without_landmarks"
destination_base = r"C:\Users\ASUS\Documents\sem 8 new\Indian_sign_lang_recognition_model\data"

# Get all subfolders in the source directory
source_folders = [folder for folder in os.listdir(source_base) if os.path.isdir(os.path.join(source_base, folder))]
source_paths = [os.path.join(source_base, folder) for folder in source_folders]

# Labels (A-Z, 0-9, NEXT, DONE, SPACE)
labels = [chr(i) for i in range(65, 91)] + [str(i) for i in range(10)] + ["NEXT", "DONE", "SPACE"]

# Create destination folder if it doesn't exist
os.makedirs(destination_base, exist_ok=True)

# Create label folders inside destination
for label in labels:
    os.makedirs(os.path.join(destination_base, label), exist_ok=True)

# Merge and rename images
for label in labels:
    image_counter = 1  # Start numbering images

    for source_path in source_paths:
        source_label_path = os.path.join(source_path, label)

        if os.path.exists(source_label_path):
            for img_name in sorted(os.listdir(source_label_path)):  # Sort to maintain order
                old_img_path = os.path.join(source_label_path, img_name)
                new_img_name = f"{image_counter}.jpg"
```

Fig.7.4. Renaming and organising collected images part-1

```python
                new_img_path = os.path.join(destination_base, label, new_img_name)

                shutil.copy2(old_img_path, new_img_path)  # Copy with new name
                image_counter += 1  # Increment counter

print("Images successfully merged and renamed in 'data' folder.")
```

Fig. 7.5. Renaming and organising collected images part-2

3. Programme for creating and saving dataset in pickle format

```python
import os
import pickle

import cv2
import mediapipe as mp

# Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)

# Dataset paths
DATA_DIR = './data'
PICKLE_DIR = r'C:\Users\ASUS\Documents\sem 8 new\Indian_sign_lang_recognition_model'
LANDMARK_IMG_DIR = r'C:\Users\ASUS\Documents\sem 8 new\Indian_sign_lang_recognition_model\Dataset\Landmark_imgs'
REJECTED_DIR = r'C:\Users\ASUS\Documents\sem 8 new\Indian_sign_lang_recognition_model\Dataset\Rejected_images'

# Create necessary directories if they don't exist
for directory in [PICKLE_DIR, LANDMARK_IMG_DIR, REJECTED_DIR]:
    os.makedirs(directory, exist_ok=True)

# Define which gestures require two hands
two_hand_gestures = {'A', 'B', 'D', 'E', 'F', 'G', 'H', 'J', 'K', 'M',
                     'N', 'P', 'Q', 'R', 'S', 'T', 'W', 'X', 'Y', 'Z'}  # Add two-hand gestures here
one_hand_gestures = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
                     'C', 'I', 'L', 'O', 'U', 'V', 'SPACE', 'NEXT', 'DONE'}  # Add single-hand gestures here

# Initialize dataset lists
data = []
labels = []
accepted_count = 0
rejected_count = 0
```

Fig. 7.6. Creating and saving dataset in pickle format part - 1

```python
# Process images
for dir_ in os.listdir(DATA_DIR):
    gesture_type = "two_hand" if dir_ in two_hand_gestures else "one_hand"

    for img_path in os.listdir(os.path.join(DATA_DIR, dir_)):
        data_aux = []
        x_ = []
        y_ = []

        img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path))
        if img is None:
            print(f"Rejected (cannot read): {dir_}/{img_path}")
            rejected_count += 1
            cv2.imwrite(os.path.join(REJECTED_DIR, img_path), img)
            continue

        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        results = hands.process(img_rgb)
        frame_with_landmarks = img.copy()

        # Count detected hands
        num_hands = len(results.multi_hand_landmarks) if results.multi_hand_landmarks else 0

        # Reject images with incorrect hand count
        if (gesture_type == "two_hand" and num_hands != 2) or num_hands == 0:
            print(f"Rejected (wrong hand count): {dir_}/{img_path}")
            rejected_count += 1
            cv2.imwrite(os.path.join(REJECTED_DIR, img_path), img)
            continue

        # Process landmarks if valid
        for hand_landmarks in results.multi_hand_landmarks:
```

Fig. 7.7. Creating and saving dataset in pickle format part - 2

```python
        # Process landmarks if valid
        for hand_landmarks in results.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                x_.append(x)
                y_.append(y)

        for hand_landmarks in results.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                data_aux.append(x - min(x_))
                data_aux.append(y - min(y_))

        # Ensure each sample has a fixed length (padding if needed)
        while len(data_aux) < 84:  # 21 landmarks * 2 coordinates * 2 hands
            data_aux.append(0.0)

        data.append(data_aux)
        labels.append(dir_)
        accepted_count += 1
        print(f"Accepted: {dir_}/{img_path}")

        # Draw landmarks on the image
        for hand_landmarks in results.multi_hand_landmarks:
            mp.solutions.drawing_utils.draw_landmarks(
                frame_with_landmarks, hand_landmarks, mp_hands.HAND_CONNECTIONS
            )

        # Save image with landmarks
        landmark_img_path = os.path.join(LANDMARK_IMG_DIR, f"{dir_}_{img_path}")
```

Fig.7.8. Creating and saving dataset in pickle format part - 3

```python
        # Save image with landmarks
        landmark_img_path = os.path.join(LANDMARK_IMG_DIR, f"{dir_}_{img_path}")
        cv2.imwrite(landmark_img_path, frame_with_landmarks)

print(f"\nTotal images accepted: {accepted_count}")
print(f"Total images rejected: {rejected_count}")

# Save dataset
pickle_path = os.path.join(PICKLE_DIR, 'data.pickle')
with open(pickle_path, 'wb') as f:
    pickle.dump( obj: {'data': data, 'labels': labels}, f)

print(f"Dataset saved at: {pickle_path}")
```

Fig. 7.9. Creating and saving dataset in pickle format part – 4

4. Programme for train model on pickle dataset

```python
import pickle
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
import numpy as np

# Load dataset
data_dict = pickle.load(open('./data.pickle', 'rb'))
data = np.asarray(data_dict['data'])
labels = np.asarray(data_dict['labels'])

# Split dataset
x_train, x_test, y_train, y_test = train_test_split( *arrays: data, labels, test_size=0.3, shuffle=True, stratify=labels)

# Train model
model = RandomForestClassifier(min_samples_leaf=30)
model.fit(x_train, y_train)

# Predict
y_predict = model.predict(x_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_predict)
precision = precision_score(y_test, y_predict, average='weighted')
recall = recall_score(y_test, y_predict, average='weighted')
f1 = f1_score(y_test, y_predict, average='weighted')

# print accuracy
print(f'Accuracy: {accuracy * 100:.2f}%')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
```

Fig. 7.10. Train model on pickle dataset part - 1

```python
# print accuracy
print(f'Accuracy: {accuracy * 100:.2f}%')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')
print('\nClassification Report:\n', classification_report(y_test, y_predict))

# Save model
with open('model.p', 'wb') as f:
    pickle.dump( obj: {'model': model}, f)
```

Fig. 7.11. Train model on pickle dataset part -2

5. C/C++ Programme for hardware to communicate with servo motor for hand tracking

```
#include <ESP32Servo.h>

Servo servoX, servoY;
int servoPinX = 14;  // X-axis Servo (ESP32-CAM GPIO 14), yellow wire
int servoPinY = 15;  // Y-axis Servo (ESP32-CAM GPIO 15), orange wire
int posX = 90, posY = 60;

void setup() {
    Serial.begin(115200);
    servoX.attach(servoPinX);
    servoY.attach(servoPinY);
    servoX.write(posX);
    servoY.write(posY);
}

void loop() {
    if (Serial.available()) {
        String data = Serial.readStringUntil('#');  // Read Serial Data
        Serial.print("Received: ");
        Serial.println(data);

        int xIndex = data.indexOf('X');
        int yIndex = data.indexOf('Y');

        if (xIndex != -1 && yIndex != -1) {
            posX = data.substring(xIndex + 1, yIndex).toInt();
            posY = data.substring(yIndex + 1).toInt();

            Serial.print("Moving to X: ");
            Serial.print(posX);
            Serial.print(", Y: ");
            Serial.println(posY);
```

Fig7.12. C/C++ Programme for ESP32-CAM part - 1

```
            servoX.write(posX);
            servoY.write(posY);
        }
    }
}
```

Fig.7.13. C/C++ Programme for ESP32-CAM part -2

6. Programme for Final TTS inference without hardware integration

```python
import pickle
import cv2
import mediapipe as mp
import numpy as np
import time
import threading
from tkinter import Tk, Label, Frame, Button
from PIL import Image, ImageTk
from gtts import gTTS
import os
from playsound import playsound
import tempfile

# ------------------------- Load Trained Model and Initialize Mediapipe -------------------------
model_dict = pickle.load(open('./model.p', 'rb'))
model = model_dict['model']

cap = cv2.VideoCapture(0)

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)

labels_dict = {
    0: '0', 1: '1', 2: '2', 3: '3', 4: '4', 5: '5', 6: '6', 7: '7', 8: '8', 9: '9',
    'A': 'A', 'B': 'B', 'C': 'C', 'D': 'D', 'E': 'E', 'F': 'F', 'G': 'G', 'H': 'H',
    'I': 'I', 'J': 'J', 'K': 'K', 'L': 'L', 'M': 'M', 'N': 'N', 'O': 'O', 'P': 'P',
    'Q': 'Q', 'R': 'R', 'S': 'S', 'T': 'T', 'U': 'U', 'V': 'V', 'W': 'W', 'X': 'X',
    'Y': 'Y', 'Z': 'Z', 'SPACE': ' ', 'DELETE': 'DELETE', 'DONE': 'DONE'
}

```

Fig.7.14. Final TTS inference without hardware part - 1

```python

# ------------------------ Global Variables for Timing, Sentence Accumulation, and Recording Control ------------------
current_gesture = None       # Currently detected gesture
gesture_start_time = None    # Time when the current gesture started being detected
sentence = ""                # Accumulated sentence
recording = False            # Controls whether sentence accumulation is active

# Threshold times (in seconds)
FIRST_GESTURE_THRESHOLD = 4     # For the first word gesture in a sentence
SUBSEQUENT_GESTURE_THRESHOLD = 4   # For subsequent word gestures
DONE_GESTURE_THRESHOLD = 4        # For finalizing the sentence when DONE is detected

# ------------------------ Text-to-Speech Function ------------------------
def speak_sentence(text):  1 usage  shubhamsony1
    try:
        with tempfile.NamedTemporaryFile(delete=False, suffix='.mp3') as temp_file:
            temp_path = temp_file.name
        tts = gTTS(text=text, lang='en')
        tts.save(temp_path)
        playsound(temp_path)
        os.remove(temp_path)
    except Exception as e:
        print("TTS Error:", e)

def tts_thread(sentence_text):  1 usage  shubhamsony1
    threading.Thread(target=speak_sentence, args=(sentence_text,)).start()

# ------------------------ Tkinter GUI Setup ------------------------
window = Tk()
window.title("Sign Gesture Recognition with TTS")

```

Fig. 7.15. Final TTS inference without hardware part - 2

```python

# Create frames for layout
video_frame = Frame(window)
video_frame.pack()

output_frame = Frame(window)
output_frame.pack()

# Label for video feed (camera output)
video_label = Label(video_frame)
video_label.pack()

# Label for current predicted gesture output
gesture_label = Label(output_frame, text="Predicted: ", font=("Helvetica", 16))
gesture_label.pack()

# Label for the sentence output
sentence_label = Label(output_frame, text="Sentence: ", font=("Helvetica", 16))
sentence_label.pack()

# Label for status (Writing or Not Writing)
status_label = Label(output_frame, text="Status: Not Writing Words", font=("Helvetica", 14))
status_label.pack(pady=5)

```

Fig.7.16. Final TTS inference without hardware part - 3

```python

# ------------------------ Button Functions ------------------------
def start_recording():  1 usage  shubhamsony1
    global recording, sentence
    recording = True
    sentence = ""  # Reset the sentence when starting
    sentence_label.config(text="Sentence: ")
    status_label.config(text="Status: Writing Words")

def stop_recording():  1 usage  shubhamsony1
    global recording
    recording = False
    status_label.config(text="Status: Not Writing Words")
    # Optionally finalize or speak the sentence on stop:
    # if sentence.strip() != "":
    #     tts_thread(sentence)

# ------------------------ Buttons ------------------------
start_button = Button(output_frame, text="Start", font=("Helvetica", 14), command=start_recording)
start_button.pack(side="left", padx=10, pady=10)

stop_button = Button(output_frame, text="Stop", font=("Helvetica", 14), command=stop_recording)
stop_button.pack(side="left", padx=10, pady=10)

```

Fig.7.17. Final TTS inference without hardware part - 4

```python
# ------------------------ Main Update Loop ------------------------
def update():  3 usages  shubhamsony1
    global current_gesture, gesture_start_time, sentence

    ret, frame = cap.read()
    if not ret:
        window.after( ms: 10, update)
        return

    H, W, _ = frame.shape
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(frame_rgb)

    data_aux = []
    x_ = []
    y_ = []
    predicted_character = ""

    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                frame, hand_landmarks, mp_hands.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style()
            )
        for hand_landmarks in results.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                x_.append(x)
                y_.append(y)
        for hand_landmarks in results.multi_hand_landmarks:
```

Fig.7.18. Final TTS inference without hardware part - 5

```python
        for hand_landmarks in results.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                data_aux.append(x - min(x_))
                data_aux.append(y - min(y_))
        while len(data_aux) < 84:
            data_aux.append(0.0)
        prediction = model.predict([np.asarray(data_aux)])
        if isinstance(prediction[0], (int, np.integer)):
            predicted_character = labels_dict.get(int(prediction[0]), "Unknown")
        else:
            try:
                predicted_character = labels_dict.get(int(prediction[0]), "Unknown")
            except:
                predicted_character = labels_dict.get(prediction[0], "Unknown")
        print(f"Predicted Raw Output: {prediction[0]}")

        x1, y1 = int(min(x_) * W) - 10, int(min(y_) * H) - 10
        x2, y2 = int(max(x_) * W) - 10, int(max(y_) * H) - 10

        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
        cv2.putText(frame, predicted_character, org: (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1.3, color: (0, 0, 0), thickness: 3, cv2.LINE_AA)

    gesture_label.config(text=f"Predicted: {predicted_character}")
```

Fig7.19. Final TTS inference without hardware part - 6

```python

    # ------------------------- Timing Logic for Sentence Accumulation -------------------------
    if recording:
        current_time = time.time()
        if predicted_character != "":
            if predicted_character == current_gesture:
                elapsed = current_time - gesture_start_time
            else:
                current_gesture = predicted_character
                gesture_start_time = current_time
                elapsed = 0

            if current_gesture == "DONE":
                threshold = DONE_GESTURE_THRESHOLD
            elif current_gesture == "DELETE":
                threshold = 5  # Hold DELETE gesture for 5 seconds to trigger
            else:
                threshold = FIRST_GESTURE_THRESHOLD if sentence == "" else SUBSEQUENT_GESTURE_THRESHOLD

            if elapsed >= threshold:
                if current_gesture == "DONE":
                    if sentence.strip() != "":
                        sentence_label.config(text=f"Sentence: {sentence}")
                        tts_thread(sentence)
                        sentence = ""
                elif current_gesture == "DELETE":
                    # Delete the last word
                    sentence = sentence[:-1]
                    sentence_label.config(text=f"Sentence: {sentence}")
                else:
                    if current_gesture == " ":
```

Fig.7.20. Final TTS inference without hardware part - 7

```
            elif current_gesture == "DELETE":
                # Delete the last word
                sentence = sentence[:-1]
                sentence_label.config(text=f"Sentence: {sentence}")
            else:
                if current_gesture == " ":
                    sentence += " "
                else:
                    sentence += current_gesture
                sentence_label.config(text=f"Sentence: {sentence}")
            current_gesture = None
            gesture_start_time = None
        else:
            current_gesture = None
            gesture_start_time = None

    img = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    imgtk = ImageTk.PhotoImage(image=img)
    video_label.imgtk = imgtk
    video_label.configure(image=imgtk)
    window.after( ms: 10, update)

# Start the GUI update loop
update()
window.mainloop()

cap.release()
cv2.destroyAllWindows()
```

Fig.7.21. Final TTS inference without hardware part -8

7. Programme for Final TTS GUI Hardware integrated inference with hardware included

```python
import pickle
import cv2
import mediapipe as mp
import numpy as np
import time
import threading
from tkinter import Tk, Label, Frame, Button
from PIL import Image, ImageTk
from gtts import gTTS
import os
from playsound import playsound
import tempfile
import serial

# ------------------------ Load Trained Model and Initialize Mediapipe ------------------------
model_dict = pickle.load(open('./model.p', 'rb'))
model = model_dict['model']

cap = cv2.VideoCapture(1)

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)

labels_dict = {
    0: '0', 1: '1', 2: '2', 3: '3', 4: '4', 5: '5', 6: '6', 7: '7', 8: '8', 9: '9',
    'A': 'A', 'B': 'B', 'C': 'C', 'D': 'D', 'E': 'E', 'F': 'F', 'G': 'G', 'H': 'H',
    'I': 'I', 'J': 'J', 'K': 'K', 'L': 'L', 'M': 'M', 'N': 'N', 'O': 'O', 'P': 'P',
    'Q': 'Q', 'R': 'R', 'S': 'S', 'T': 'T', 'U': 'U', 'V': 'V', 'W': 'W', 'X': 'X',
    'Y': 'Y', 'Z': 'Z', 'SPACE': ' ', 'DELETE': 'DELETE', 'DONE': 'DONE'
}
```

Fig.7.22. Final TTS GUI Hardware integrated inference part - 1

```python

# ------------------------ Global Variables for Timing, Sentence Accumulation, and Recording Control -----------------
current_gesture = None       # Currently detected gesture
gesture_start_time = None    # Time when the current gesture started being detected
sentence = ""                # Accumulated sentence
recording = False            # Controls whether sentence accumulation is active

# Threshold times (in seconds)
FIRST_GESTURE_THRESHOLD = 4     # For the first word gesture in a sentence
SUBSEQUENT_GESTURE_THRESHOLD = 4   # For subsequent word gestures
DONE_GESTURE_THRESHOLD = 4        # For finalizing the sentence when DONE is detected

# ------------------------ Text-to-Speech Function ------------------------
def speak_sentence(text):  1 usage  shubhamsony1
    try:
        with tempfile.NamedTemporaryFile(delete=False, suffix='.mp3') as temp_file:
            temp_path = temp_file.name
        tts = gTTS(text=text, lang='en')
        tts.save(temp_path)
        playsound(temp_path)
        os.remove(temp_path)
    except Exception as e:
        print("TTS Error:", e)

def tts_thread(sentence_text):  1 usage  shubhamsony1
    threading.Thread(target=speak_sentence, args=(sentence_text,)).start()

# ------------------------ Tkinter GUI Setup ------------------------
window = Tk()
window.title("Sign Gesture Recognition with TTS")

```

Fig.7.23. Final TTS GUI Hardware integrated inference part - 2

```python

# Create frames for layout
video_frame = Frame(window)
video_frame.pack()

output_frame = Frame(window)
output_frame.pack()

# Label for video feed (camera output)
video_label = Label(video_frame)
video_label.pack()

# Label for current predicted gesture output
gesture_label = Label(output_frame, text="Predicted: ", font=("Helvetica", 16))
gesture_label.pack()

# Label for the sentence output
sentence_label = Label(output_frame, text="Sentence: ", font=("Helvetica", 16))
sentence_label.pack()

# Label for status (Writing or Not Writing)
status_label = Label(output_frame, text="Status: Not Writing Words", font=("Helvetica", 14))
status_label.pack(pady=5)

# ------------------------ Button Functions ------------------------
def start_recording():  1 usage  shubhamsony1
    global recording, sentence
    recording = True
    sentence = ""  # Reset the sentence when starting
    sentence_label.config(text="Sentence: ")
    status_label.config(text="Status: Writing Words")

```

Fig. 7.24. Final TTS GUI Hardware integrated inference part - 3

```python

def stop_recording():  1 usage  shubhamsony1
    global recording
    recording = False
    status_label.config(text="Status: Not Writing Words")
    # Optionally finalize or speak the sentence on stop:
    # if sentence.strip() != "":
    #     tts_thread(sentence)

# ------------------------ Buttons ------------------------
start_button = Button(output_frame, text="Start", font=("Helvetica", 14), command=start_recording)
start_button.pack(side="left", padx=10, pady=10)

stop_button = Button(output_frame, text="Stop", font=("Helvetica", 14), command=stop_recording)
stop_button.pack(side="left", padx=10, pady=10)

# ------------------------ Servo Control Variables ------------------------


ws, hs = 1280, 720
port = "COM5"  # Change to your ESP32 COM port
ser = serial.Serial(port, baudrate: 115200, timeout=1)
time.sleep(2)

current_servoX = 90
current_servoY = 60
step_size = 1
hand_detected_frames = 0
patrol_mode_active = False
phase = 0
y_adjustments = [10, -10, -10, 10]

```

Fig.7.25. Final TTS GUI Hardware integrated inference part - 4

```python

# ------------------------ Main Update Loop ------------------------
def update():  3 usages  shubhamsony1
    global current_gesture, gesture_start_time, sentence
    global current_servoX, current_servoY, step_size, hand_detected_frames
    global patrol_mode_active, phase

    ret, frame = cap.read()
    if not ret:
        window.after( ms: 10, update)
        return

    frame = cv2.flip(frame,  flipCode: 1)
    H, W, _ = frame.shape
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(frame_rgb)

    data_aux = []
    x_ = []
    y_ = []
    predicted_character = ""

    if results.multi_hand_landmarks:
        patrol_mode_active = False
        hand_detected_frames += 1

        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                frame, hand_landmarks, mp_hands.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style()
            )
```

Fig. 7.26. Final TTS GUI Hardware integrated inference part - 5

```python

        for hand_landmarks in results.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                x_.append(x)
                y_.append(y)
        for hand_landmarks in results.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                data_aux.append(x - min(x_))
                data_aux.append(y - min(y_))
        while len(data_aux) < 84:
            data_aux.append(0.0)

        prediction = model.predict([np.asarray(data_aux)])
        if isinstance(prediction[0], (int, np.integer)):
            predicted_character = labels_dict.get(int(prediction[0]), "Unknown")
        else:
            try:
                predicted_character = labels_dict.get(int(prediction[0]), "Unknown")
            except:
                predicted_character = labels_dict.get(prediction[0], "Unknown")

        gesture_label.config(text=f"Predicted: {predicted_character}")
        print(f"Predicted Raw Output: {prediction[0]}")

```

Fig. 7.27. Final TTS GUI Hardware integrated inference part - 6

```python
        # Bounding box for gesture prediction
        x1, y1 = int(min(x_) * W) - 10, int(min(y_) * H) - 10
        x2, y2 = int(max(x_) * W) - 10, int(max(y_) * H) - 10
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
        cv2.putText(frame, predicted_character, org: (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1.3, color: (0, 0, 0), thickness: 3, cv2.LINE_AA)

        # ------------------------ Servo Logic ------------------------
        x_min, y_min = int(min(x_) * ws), int(min(y_) * hs)
        x_max, y_max = int(max(x_) * ws), int(max(y_) * hs)
        x_min -= 20
        y_min -= 20
        x_max += 20
        y_max += 20

        hand_cx = (x_min + x_max) // 2
        hand_cy = (y_min + y_max) // 2
        camera_cx, camera_cy = ws // 2, hs // 2

        target_servoX = int(np.interp(hand_cx, xp: [0, ws], fp: [0, 180]))
        target_servoY = int(np.interp(hand_cy, xp: [0, hs], fp: [0, 180]))

        inside_x = x_min <= camera_cx <= x_max
        inside_y = y_min <= camera_cy <= y_max

        if not (inside_x and inside_y):
            if not inside_x:
                if abs(target_servoX - current_servoX) > step_size:
                    current_servoX += step_size if target_servoX > current_servoX else -step_size
                else:
```

Fig. 7.28. Final TTS GUI Hardware integrated inference part - 7

```
                    current_servoX = target_servoX

            if not inside_y:
                if abs(target_servoY - current_servoY) > step_size:
                    current_servoY += step_size if target_servoY > current_servoY else -step_size
                else:
                    current_servoY = target_servoY

        command = f"X{current_servoX}Y{current_servoY}#"
        ser.write(command.encode())
        print(f"Sent: {command}")

    else:
        hand_detected_frames = 0

        if not patrol_mode_active:
            if abs(current_servoX - 90) > step_size:
                current_servoX += step_size if 90 > current_servoX else -step_size
            else:
                current_servoX = 90
            if abs(current_servoY - 90) > step_size:
                current_servoY += step_size if 90 > current_servoY else -step_size
            else:
                current_servoY = 90

            if current_servoX == 90 and current_servoY == 90:
                patrol_mode_active = True
                current_servoX = 20
                phase = 0
```

Fig.7.29. Final TTS GUI Hardware integrated inference part - 8

```
        if patrol_mode_active:
            if phase in [0, 2]:
                current_servoX += step_size
                if current_servoX >= 160:
                    current_servoX = 160
                    if phase == 2:
                        current_servoY += y_adjustments[phase]
                    phase = (phase + 1) % 4
            else:
                current_servoX -= step_size
                if current_servoX <= 20:
                    current_servoX = 20
                    current_servoY += y_adjustments[phase]
                    phase = (phase + 1) % 4

            if phase == 0 and current_servoX == 20:
                current_servoY += y_adjustments[phase]

        command = f"X{current_servoX}Y{current_servoY}#"
        ser.write(command.encode())
        print(f"Patrol Mode: {command}")

    # ------------------------- Sentence Accumulation Logic -------------------------
    if recording:
        current_time = time.time()
        if predicted_character != "":
            if predicted_character == current_gesture:
                elapsed = current_time - gesture_start_time
            else:
                current_gesture = predicted_character
```

Fig. 7.30. Final TTS GUI Hardware integrated inference part - 9

```
                gesture_start_time = current_time
                elapsed = 0

            if current_gesture == "DONE":
                threshold = DONE_GESTURE_THRESHOLD
            elif current_gesture == "DELETE":
                threshold = 5
            else:
                threshold = FIRST_GESTURE_THRESHOLD if sentence == "" else SUBSEQUENT_GESTURE_THRESHOLD

            if elapsed >= threshold:
                if current_gesture == "DONE":
                    if sentence.strip() != "":
                        sentence_label.config(text=f"Sentence: {sentence}")
                        tts_thread(sentence)
                        sentence = ""
                elif current_gesture == "DELETE":
                    sentence = sentence[:-1]
                    sentence_label.config(text=f"Sentence: {sentence}")
                else:
                    if current_gesture == " ":
                        sentence += " "
                    else:
                        sentence += current_gesture
                    sentence_label.config(text=f"Sentence: {sentence}")
                current_gesture = None
                gesture_start_time = None
        else:
            current_gesture = None
            gesture_start_time = None
```

Fig. 7.31. Final TTS GUI Hardware integrated inference part - 10

```
    img = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    imgtk = ImageTk.PhotoImage(image=img)
    video_label.imgtk = imgtk
    video_label.configure(image=imgtk)
    window.after( ms: 10, update)

# Start the GUI update loop
update()
window.mainloop()

cap.release()
cv2.destroyAllWindows()
```

Fig.7.32. Final TTS GUI Hardware integrated inference part – 11

Git Hub Repo Link: -https://github.com/shubhamsony1/Indian_sign_lang_recognition_model

CHAPTER 9

REFERENCE

[1] WHO, '"Blindness and vision impairment." World Health Organization: WHO (2021).' Accessed: Apr. 03, 2025.

[2] 'National Programme for the Prevention & Control of Deafness (NPPCD): National Health Mission'. Accessed: Apr. 08, 2025.

[3] J. Shajeena, R. M. Shiny, M. Mary Vespa, and R. Kavitha, 'Sign Language Translation and Voice Impairment Support System using Deep Learning', in *Proceedings of the 5th International Conference on Data Intelligence and Cognitive Informatics, ICDICI 2024*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 558–563. doi: 10.1109/ICDICI62993.2024.10810957.

[4] T. Bharti, S. Singh, V. P. Chaubey, S. Gupta, S. Sharma, and S. Gochhait, 'Next-Gen Assistive Technology: AI Applications for Deaf-Mute and Blind Communities', in *2024 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies, 3ICT 2024*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 359–365. doi: 10.1109/3ICT64318.2024.10823653.

[5] Suraksha Devi and Suman Deb, *Low cost tangible glove for translating sign gestures to speech and text in Hindi language*. IEEE, 2017.

[6] Geetha M and Harikrishnan R, 'A Vision Based Dynamic Gesture Recognition of Indian Sign Language on Kinect based Depth Images'.

[7] G. Sánchez-Brizuela, A. Cisnal, E. de la Fuente-López, J. C. Fraile, and J. Pérez-Turiel, 'Lightweight real-time hand segmentation leveraging MediaPipe landmark detection', *Virtual Real*, vol. 27, no. 4, pp. 3125–3132, Dec. 2023, doi: 10.1007/S10055-023-00858-0/FIGURES/7.

[8] C. Lugaresi *et al.*, 'MediaPipe: A Framework for Building Perception Pipelines', Jun. 2019, Accessed: Apr. 05, 2025. [Online]. Available: https://arxiv.org/abs/1906.08172v1

[9] K. Dokic, ‘Microcontrollers on the edge – is esp32 with camera ready for machine learning?’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12119 LNCS, pp. 213–220, 2020, doi: 10.1007/978-3-030-51935-3_23/FIGURES/4.

[10] A. B., K. Krishi, M. M., M. Daaniyaal, and A. H. S., ‘Hand gesture recognition using machine learning algorithms’, *Computer Science and Information Technologies*, vol. 1, no. 3, pp. 116–120, Nov. 2020, doi: 10.11591/CSIT.V1I3.P116-120.

[11] E. Casanova *et al.*, ‘SC-GlowTTS: an Efficient Zero-Shot Multi-Speaker Text-To-Speech Model’, *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 5, pp. 3546–3550, Apr. 2021, doi: 10.21437/Interspeech.2021-1774.

[12] H. Guo, S. Zhang, F. K. Soong, L. He, and L. Xie, ‘Conversational End-to-End TTS for Voice Agents’, *2021 IEEE Spoken Language Technology Workshop, SLT 2021 - Proceedings*, pp. 403–409, Jan. 2021, doi: 10.1109/SLT48900.2021.9383460.

[13] R. K. Veluri, S. R. Sree, A. Vanathi, G. Aparna, and S. P. Vaidya, ‘Hand Gesture Mapping Using MediaPipe Algorithm’, *Lecture Notes in Electrical Engineering*, vol. 844, pp. 597–614, 2022, doi: 10.1007/978-981-16-8862-1_39.

[14] R. C. Joshi, N. Singh, A. K. Sharma, R. Burget, and M. K. Dutta, ‘AI-SenseVision: A Low-Cost Artificial-Intelligence-Based Robust and Real-Time Assistance for Visually Impaired People’, *IEEE Trans Hum Mach Syst*, vol. 54, no. 3, pp. 325–336, Jun. 2024, doi: 10.1109/THMS.2024.3375655.

[15] A. Chaudhary, S. Kolhe, and R. Kamal, ‘An improved random forest classifier for multi-class classification’, *Information Processing in Agriculture*, vol. 3, no. 4, pp. 215–222, Dec. 2016, doi: 10.1016/J.INPA.2016.08.002.

[16] A. Parmar, R. Katariya, and V. Patel, ‘A Review on Random Forest: An Ensemble Classifier’, *Lecture Notes on Data Engineering and Communications Technologies*, vol. 26, pp. 758–763, 2019, doi: 10.1007/978-3-030-03146-6_86.

[17] C. Sharma, S. Sharma, M. Kumar, and A. Sodhi, 'Early Stroke Prediction Using Machine Learning', in *2022 International Conference on Decision Aid Sciences and Applications, DASA 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 890–894. doi: 10.1109/DASA54658.2022.9765307.

[18] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, 'How Many Trees in a Random Forest?', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7376 LNAI, pp. 154–168, 2012, doi: 10.1007/978-3-642-31537-4_13.

[19] V. P. Chaubey, A. Sharma, T. Sharma, S. Sharma, and A. Kumar, 'Link Prediction for Social Network Analysis Using Random Forest and XG-Boost Algorithm', *Proceedings of 2023 2nd International Conference on Informatics, ICI 2023*, 2023, doi: 10.1109/ICI60088.2023.10421655.

[20] A. Sharma, Sameer, T. Pal, and V. Jaiswal, 'Decision support algorithms for data analysis', Mar. 18, 2021, *Nova Science Publishers, Inc.* Accessed: Apr. 04, 2025. [Online]. Available: https://cris.bgu.ac.il/en/publications/decision-support-algorithms-for-data-analysis

[21] Jinsu Kunjumon and Dr. Rajesh Kannan Megalingam, *Hand Gesture Recognition System For Translating Indian Sign Language Into Text And Speech*. IEEE, 2019.

[22] N. V. Y. Lankadasu, D. B. Pesarlanka, A. Sharma, and S. Sharma, 'Security Aspects of Blockchain Technology', *https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/979-8-3693-2869-9.ch014*, pp. 259–277, doi: 10.4018/979-8-3693-2869-9.CH014.

[23] A. Trilla and F. Alias, 'Sentence-based sentiment analysis for expressive text-to-speech', *IEEE Trans Audio Speech Lang Process*, vol. 21, no. 2, pp. 223–233, 2013, doi: 10.1109/TASL.2012.2217129.

[24] Crocioni Giulia, Di Vece Chiara, and Hananeh Esmailbeigi, *Glove for Speech Recognition*. IEEE, 2019.

[25] I. J. Bush, R. Abiyev, and M. Arslan, 'Impact of machine learning techniques on hand gesture recognition', *Journal of Intelligent and Fuzzy Systems*, vol. 37, no. 3, pp. 4241–4252, 2019, doi: 10.3233/JIFS-190353.

[26] A. Sharma, A. Gupta, and V. Jaiswal, ‘Solving Image Processing Critical Problems Using Machine Learning’, *Studies in Big Data*, vol. 82, pp. 213–248, 2021, doi: 10.1007/978-981-15-9492-2_11.

[27] A. Sharma, V. Guleria, and V. Jaiswal, ‘The Future of Blockchain Technology, Recent Advancement and Challenges’, *Studies in Big Data*, vol. 105, pp. 329–349, 2022, doi: 10.1007/978-3-030-95419-2_15.

[28] Y. Kwon and D. Kim, ‘Real-Time Workout Posture Correction using OpenCV and MediaPipe’, *The Journal of Korean Institute of Information Technology*, vol. 20, no. 1, pp. 199–208, Jan. 2022, doi: 10.14801/JKIIT.2022.20.1.199.

[29] S. Kumar, K. Sharma, G. Raj, D. Datta, and A. Ghosh, ‘Arduino and ESP32-CAM-Based Automatic Touchless Attendance System’, *Lecture Notes in Electrical Engineering*, vol. 851, pp. 135–144, 2022, doi: 10.1007/978-981-16-9154-6_14.

[30] B. Das and K. K. Halder, ‘Face Recognition Using ESP32-Cam for Real-Time Tracking and Monitoring’, *2024 International Conference on Advances in Computing, Communication, Electrical, and Smart Systems: Innovation for Sustainability, iCACCESS 2024*, 2024, doi: 10.1109/ICACCESS61735.2024.10499606.

Checklist for Dissertation Supervisor

Name: _____ UID: _____ Domain: ____

Registration No: _____Name of student:_____

Title of Dissertation: _____

- ☐ Front pages are as per the format.
- ☐ Topic on the PAC form and title page is same.
- ☐ Front page numbers are in roman and for report, it is like 1, 2,3…….
- ☐ TOC, List of Figures, etc. are matching with the actual page numbers in the report.
- ☐ Font, Font Size, Margins, line Spacing, Alignment, etc. are as per the guidelines.
- ☐ Color prints are used for images and implementation snapshots. o Captions and citations are provided for all the figures, tables etc. and are numbered and center aligned.
- ☐ All the equations used in the report are numbered.
- ☐ Citations are provided for all the references.
- ☐ Objectives are clearly defined.
- ☐ Minimum total number of pages of report is 50. o Minimum references in report are 30.

Here by, I declare that I had verified the above-mentioned points in the final dissertation report.

Signature of Supervisor with UID

RESEARCH PAPER (STATUS)

Outlook

4th IEEE International Conference on Sensors and Related Networks - Special Focus on Digital Healthcare : Submission (304) has been created.

From Microsoft CMT <noreply@msr-cmt.org>
Date Tue 15/04/2025 16:05
To shubhamsony20112002@gmail.com <shubhamsony20112002@gmail.com>

Hello,

The following submission has been created.

Track Name: SENNET2025

Paper ID: 304

Paper Title: Indian sign language to voice using ESP32_Cam for hand tracking and Media Pipe for hand gesture detection

Abstract:
Communication gaps experienced by people with hearing and speech disabilities continue to be a major issue in the modern world, particularly when communicating with people who are not familiar with sign language To bridge this gap, this study introduces a real-time Indian Sign Language (ISL) to Voice translation system based on an ESP32-CAM module for capturing hand gestures and Media-Pipe for precise hand landmark detection and gesture recognition. The system identifies ISL alphabets and numbers using effective hand tracking and gesture segmentation, providing smooth conversion of sign language to text as well as synthesized speech output. Media-Pipe's hand tracking improves the precision of gesture recognition by capturing 21 critical hand landmarks while a light machine learning model determines the gestures. The identified text is then converted to voice with the help of text-to-speech (TTS) synthesis, enabling clear communication for the hearing and speech impaired. Differing from conventional systems depending on heavy hardware or being confined to English speech, this project focuses on an embedded, cost-effective solution via ESP32-CAM, thereby being portable. In addition, the flexibility in the system will enable future enhancement for multilingual, even regional languages such as Malayalam, to extend inclusivity. This paper describes the architecture, implementation, and experimental performance of the system, showing its efficiency and accuracy in real-time ISL gesture recognition and voice generation. The system proposed here has great potential in filling the communication gaps, enabling independence and social inclusion of hearing and speech disabled individuals.

Created on: Tue, 15 Apr 2025 10:35:02 GMT

Last Modified: Tue, 15 Apr 2025 10:35:02 GMT

Authors:

- shubhamsony20112002@gmail.com
- arjunlande07@gmail.com
- rukna2003@gmail.com
- ajaysharmadharmani@gmail.com (Primary)
- shamneesh.sharma@gmail.com

Primary Subject Area: AI/ML and Data Analytics in Sensor Informatics

Secondary Subject Areas: Not Entered

Submission Files:
Hand Gesture Manuscript..pdf (696 Kb, Tue, 15 Apr 2025 10:34:05 GMT)

Submission Questions Response: Not Entered

Thanks,
CMT team.

To stop receiving conference emails, you can check the 'Do not send me conference email' box from your User Profile.

Microsoft respects your privacy. To learn more, please read our Privacy Statement.

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

PLAGIARISM (STATUS)

- **Project Report Plagiarism**

ABSTRACT

Communication gaps experienced by people with speech and hearing disabilities continue to be a major issue in the modern world, particularly when communicating with people who does not know sign language to bridge this gap, this study introduces a real-time Indian Sign Language (ISL) to Voice translation system based on an ESP32-CAM module for capturing hand gestures and Media-Pipe for precise hand landmark detection and gesture recognition. The system identifies ISL alphabets and numbers using effective hand tracking and gesture segmentation, providing smooth conversion of sign language to text as well as synthesized speech output. Media-Pipe's hand tracking improves the precision of gesture recognition by capturing 21 critical hand landmarks, while a light machine learning model determines the gestures. The identified text is then converted to voice with the help of text-to-speech (TTS) synthesis, enabling clear communication for the speech and hearing impaired. Differing from conventional systems depending on heavy hardware or being confined to English speech, this project focuses on an embedded, cost-effective solution via ESP32-CAM, thereby being portable. In addition, the flexibility in the system will enable future enhancement for multilingual even regional languages such as Malayalam, to extend inclusivity. This paper describes the implementation, architecture and experimental performance of the system, showing its efficiency and accuracy in real-time ISL gesture recognition and voice generation. The system proposed here has great potential in filling the communication gaps, enabling independence and social inclusion of hearing and speech disabled individuals.

Keywords: Indian Sign Language (ISL), ESP32-CAM, Media-Pipe, Hand Gesture Recognition, Text-to-Speech (TTS), Machine Learning, Random Forest.

- **Normal Plagiarism**

turnitin

Similarity Report ID: oid:3618:93200362

PAPER NAME

0427150200

WORD COUNT	CHARACTER COUNT
6203 Words	37350 Characters
PAGE COUNT	FILE SIZE
67 Pages	6.3MB
SUBMISSION DATE	REPORT DATE
Apr 27, 2025 10:02 PM UTC	Apr 27, 2025 10:03 PM UTC

8% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 3% Internet database
- 4% Publications database
- Crossref database
- Crossref Posted Content database
- 6% Submitted Works database

Summary

- **AI Plagiarism**

0427150200

Turnitin

Document Details

Submission ID
trn:oid:::3618:93200362

Submission Date
Apr 27, 2025, 10:02 PM UTC

Download Date
Apr 27, 2025, 10:04 PM UTC

File Name
0427150200_1_0_0_0_1_0_0_.docx

File Size
6.3 MB

67 Pages

6,203 Words

37,350 Characters

turnitin Page 1 of 69 - Cover Page Submission ID trn:oid:::3618:93200362

0% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups

0 AI-generated only 0%
Likely AI-generated text from a large-language model.

0 AI-generated text that was AI-paraphrased 0%
Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.