iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** III **Month of publication:** March 2026

DOI: https://doi.org/10.22214/ijraset.2026.78173

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue III Mar 2026- Available at www.ijraset.com

Deep Learning-Based Sign-To-Speech Interpretation

Mrs. J. Madhuri[1], Mrs. M. Mounika[2]
[1]*Asst. Professor, CSE Department, PBR Visvodaya Institute of Technology and Science, Kavali*
[2]*M.Tech PG Scholar, CSE, PBR Visvodaya Institute of Technology and Science, Kavali*

Abstract: ***Sign language detection system designed to assist individuals with hearing and speech impairments by enabling seamless communication. The study explores the application of advanced computer vision and deep learning techniques to interpret and translate sign language gestures into text or speech. By leveraging Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), the system effectively recognizes and understands hand movements, finger positions, and complex gestures. These neural networks are trained on a large dataset containing diverse sign language examples, allowing the system to improve accuracy and adapt to various users.***

To enhance adaptability, transfer learning is implemented, enabling the system to learn new sign languages without requiring extensive new datasets. This approach ensures that the model can support multiple sign languages and dialects, making it a versatile tool for global accessibility. Real-time processing is a crucial aspect of the system, achieved through the use of lightweight client-side models that operate efficiently on edge devices such as smartphones, tablets, and embedded systems. This reduces dependency on cloud services, ensuring low latency and fast communication, which is essential for practical usage in everyday interactions. Additionally, the system incorporates Media Pipe for hand tracking and OpenCV for real-time video analysis, ensuring robustness under varying lighting conditions, backgrounds, and hand orientations. The goal of this research is to develop an inclusive communication tool that bridges the gap between sign language users and those unfamiliar with sign language. By making sign language translation more accessible, the project promotes social inclusion, educational opportunities, and workplace integration for the deaf and hard-of-hearing community. This work lays the foundation for future advancements, including gesture-based AI assistants, smart wearable devices, and multilingual sign language recognition, further enhancing accessibility worldwide.

Keywords: Convolutional Neural Networks (CNNs)); Long Short-Term Memory (LSTM) networks; Numpy; Mediapipe; OpenCV (Open-Source Computer Vision Library).

I. INTRODUCTION

This project aims to develop a real-time sign language detection system using Long Short-Term Memory (LSTM) neural networks, a specialized form of Recurrent Neural Networks (RNNs) designed for processing sequential data. Given that sign language consists of continuous hand gestures and movements over time, LSTM models are well-suited for learning the temporal dependencies between consecutive frames in video-based gesture recognition. The system will utilize computer vision techniques, deep learning, and transfer learning to accurately detect, classify, and translate sign language gestures into text.

The successful implementation of this project will provide an accessible and scalable communication tool that bridges the gap between sign language users and non- signers. By leveraging LSTM-based neural networks, we aim to create an AI-powered assistive technology that enhances inclusion and accessibility for the deaf and hard-of- hearing community.

II. LITERATURE SURVEY

To ensure a comprehensive literature review, it is crucial to identify relevant keywords related to sign language detection. These keywords help in filtering research articles and studies that focus on AI-driven gesture recognition. Key terms include:

1) Sign Language Recognition
2) Gesture Recognition
3) Hand Tracking
4) Computer Vision
5) Deep Learning and Neural Networks
6) LSTM (Long Short-Term Memory) Networks

©IJRASET: All Rights are Reserved

7) Real-time Video Processing
8) Media Pipe-based Landmark Detection
9) CNN-LSTM Hybrid Models
10) Transfer Learning for Gesture Recognition

These keywords were used to guide searches across academic research papers, books, and online resources focusing on deep learning models (CNNs, LSTMs), OpenCV for image processing, and real-time AI implementations for sign language recognition.

By applying filters such as publication date, peer-reviewed sources, and domain- specific studies, the research was refined to focus on recent advancements in real-time sign language detection.[6] Specific emphasis was placed on Media Pipe for hand tracking, CNNs for gesture classification, and LSTMs for sequential processing.

Recent studies emphasize the importance of transfer learning in adapting sign language detection models to different languages and dialects. Many state-of-the-art models use pre-trained deep learning architectures to reduce training time and dataset requirements while maintaining high accuracy.

Additionally, research has explored error detection mechanisms in sign language models to improve gesture classification accuracy, particularly in challenging conditions such as lighting variations, hand occlusions, and background noise. Some studies have highlighted the role of depth-sensing cameras in improving gesture tracking accuracy by capturing 3D hand movements.

III. EXISTING SYSTEM

Most existing sign language detection systems use Convolutional Neural Networks (CNNs) for recognizing hand shapes and gestures. While CNNs are highly effective in static gesture classification, they struggle with dynamic and continuous sign language recognition, which involves sequential hand movements and complex gestures. To address this, some systems integrate Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks, which are designed to handle sequential data. However, these models require large datasets and extensive training, making their deployment challenging in real-world applications.

Several mobile applications have been developed to recognize sign language gestures using smartphone cameras. These apps analyze video input and provide real- time translation of sign language into text or speech. While mobile apps offer convenience and accessibility, their accuracy is often dependent on camera quality, lighting conditions, and processing power. Additionally, most applications are limited to a small set of predefined gestures and may not support regional or dialectal variations in sign language.

The system will involve capturing real-time video input using Open CV and MediaPipe, followed by preprocessing the data to extract essential hand landmarks. These landmark features will serve as the foundation for gesture classification. Unlike traditional CNN-based models that focus on static hand gestures, this system integrates LSTM (Long Short-Term Memory) networks to recognize sequential movements, making it capable of interpreting dynamic sign language gestures more effectively.

IV. PROPOSED SYSTEM

The proposed system aims to develop an advanced real-time sign language detection model by leveraging computer vision, deep learning, and real-time processing techniques. This system is designed to overcome the limitations of existing methods, ensuring higher accuracy, adaptability, and real-world usability.

The system will involve capturing real-time video input using OpenCV and MediaPipe, followed by preprocessing the data to extract essential hand landmarks. These landmark features will serve as the foundation for gesture classification. Unlike traditional CNN-based models that focus on static hand gestures, this system integrates LSTM (Long Short-Term Memory) networks to recognize sequential movements, making it capable of interpreting dynamic sign language gestures more effectively.

To improve robustness, the system will utilize data augmentation techniques such as rotation, scaling, and background adaptation to ensure better generalization across different lighting conditions and hand variations. Additionally, transfer learning will be implemented to expand the system's sign language vocabulary without requiring large amounts of new data.

V. FEASIBILITY STUDY

The proposed system integrates computer vision, machine learning, and real- time processing to recognize and interpret sign language gestures efficiently. Unlike traditional CNN-based models, which are primarily designed for recognizing static gestures, this system incorporates LSTM (Long Short-Term Memory) networks to process sequential gestures dynamically. This allows the system to understand the fluid motion of sign language, improving its ability to recognize complex sentence formations rather than isolated words.

The system requires a camera-equipped device (PC, laptop, or smartphone) with sufficient computing power to handle real-time video processing. The software stack includes:

1) Python as the primary programming language.
2) TensorFlow/Keras for deep learning model development.
3) OpenCV for video frame processing and image manipulation.
4) Media Pipe for efficient hand tracking and landmark detection.
5) Tkinter for creating an intuitive GUI (Graphical User Interface).

The development and deployment of the system will follow a structured roadmap:

1) Data Collection & Preprocessing: Capturing real-world sign language gestures and applying augmentation.
2) Model Development & Training: Implementing CNN-LSTM networks for sign classification.
3) Real-time Testing & Optimization: Ensuring gesture detection works under different lighting and backgrounds.
4) User Interface Development: Creating a Tkinter-based GUI for real-time interaction.
5) Deployment & Evaluation: Conducting user testing, refining the model, and ensuring accessibility ***compliance.***

VI. SYSTEM DESIGN

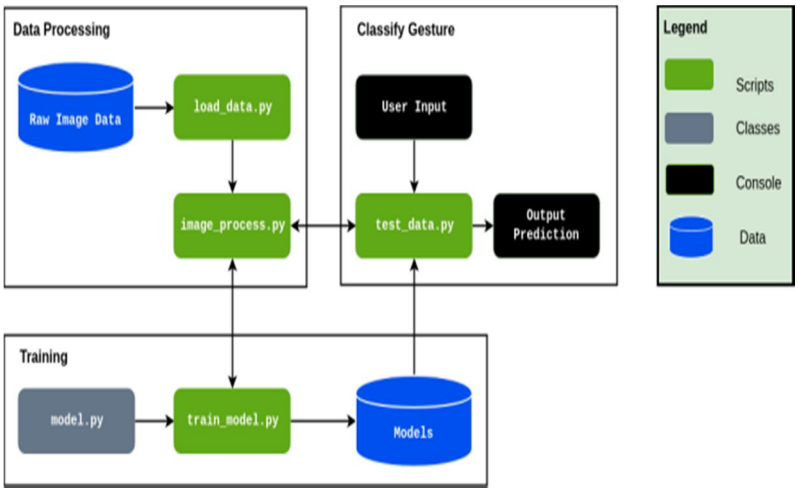
A. Requirements

The system should be able to accurately detect and classify sign language gestures. This includes:

1) Capturing hand movements, finger positions, and facial expressions.
2) Differentiating between similar gestures with high precision.
3) Recognizing dynamic gestures that involve motion sequences.
4) Supporting a wide range of sign languages, including regional and customized gestures..

Designing a sign language detection system requires a structured approach integrating computer vision, deep learning, and real-time processing. The system starts with an input module, where a camera captures video streams of hand gestures. Preprocessing techniques such as noise reduction, background segmentation, and image normalization enhance data quality. The gesture recognition module leverages CNNs for spatial feature extraction and LSTMs for sequential gesture processing, ensuring accurate interpretation of dynamic sign language gestures. OpenCV and MediaPipe handle hand tracking and landmark detection, providing real-time feature extraction.

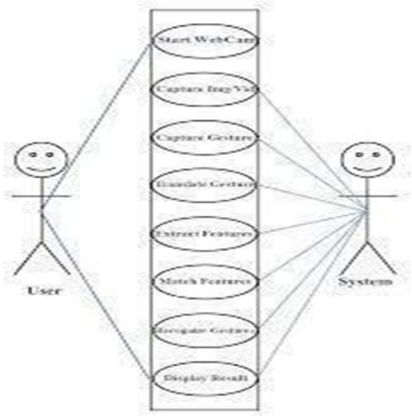
The architecture of a sign language detection system consists of multiple interconnected components that work together to capture, process, and recognize sign language gestures with high accuracy. The system follows a structured pipeline integrating computer vision, deep learning, and real-time processing to ensure efficient sign recognition.



The use cases identified in the diagram encompass the key functionalities offered by the sign language detection system. These include "Capture Sign Language Gesture," where the user utilizes input devices such as cameras or wearable sensors to capture sign language gestures.

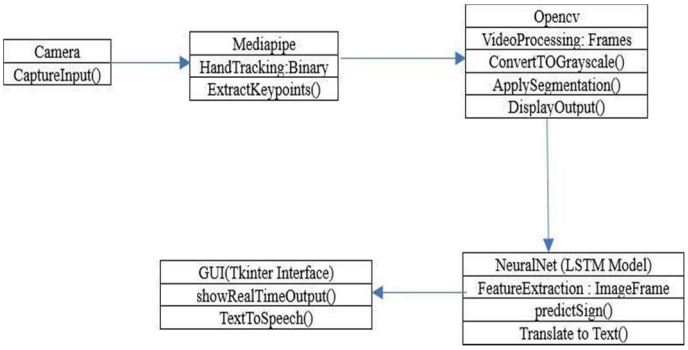
.

The "Interpret Sign Language Gesture" use case involves the system analyzing the captured gestures and interpreting their meaning using machine learning algorithms or pattern recognition techniques.

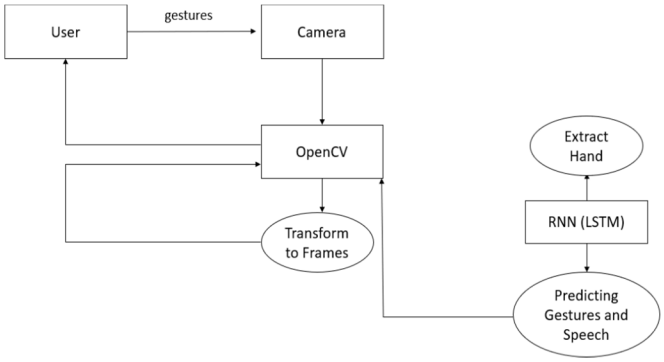


Use Case Diagram

Class Diagram



Sequence Diagram



Activity Diagram

VII. SYSTEM IMPLEMENTATION

A. Overview of Python

- Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, and Unix shell and other scripting languages.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue III Mar 2026- Available at www.ijraset.com

- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

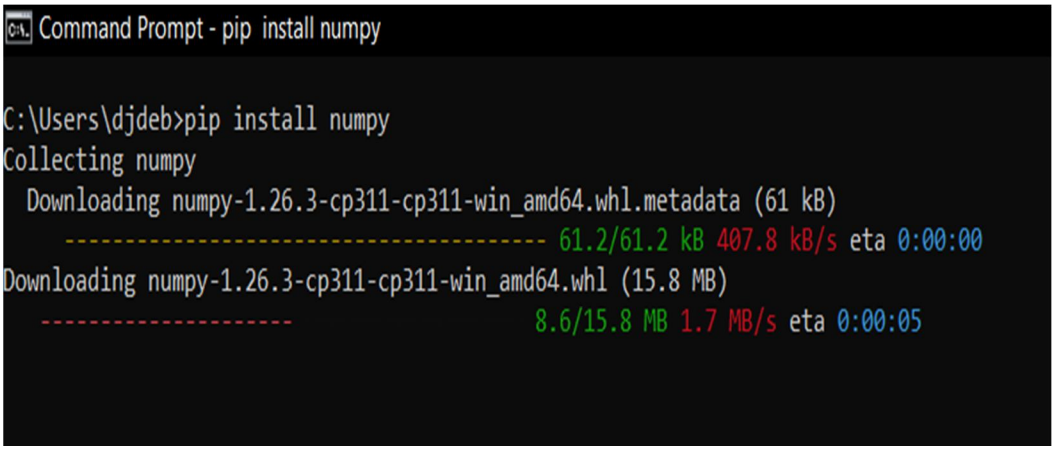
B. Python Variables

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

C. Python Libraries

1) NumPy
2) CV2
3) Mediapipe
4) Scikit-Learn

a) Numpy



Code Snippet

b) OpenCV Module

- Description: OpenCV (Open-Source Computer Vision Library) is an open source computer vision and machine learning software library.
- Purpose: Used for real-time image acquisition from the webcam.
- Features: Offers a wide range of functionalities for image processing, video analysis, object detection, and more.

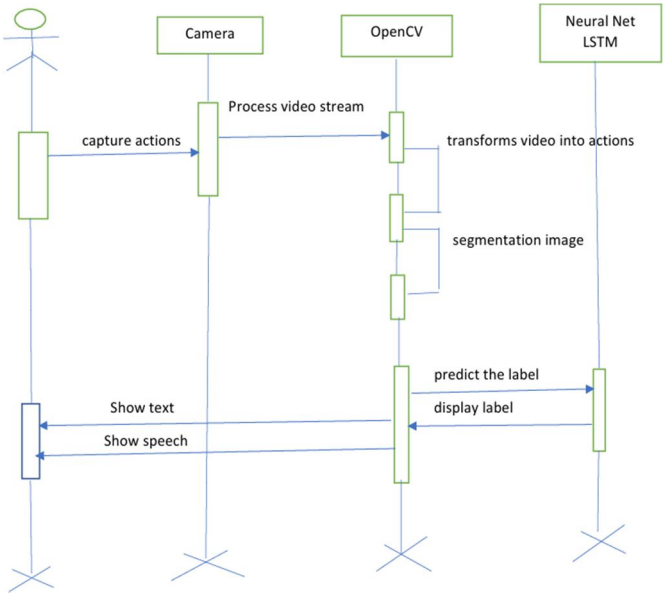
c) Scikit-learn

It provides a wide range of tools for building machine learning models, including algorithms for classification, regression, clustering, dimensionality reduction, and more. With scikit-learn, you can preprocess data, train models, perform cross-validation, and evaluate model performance. It's widely used in both academia and industry for various machine learning tasks.

d) Media Pipe

Media pipe is a powerful cross-platform framework developed by Google that enables the building of machine learning-based solutions for various perceptual tasks, including facial recognition, hand tracking, pose estimation, object detection, and more. It provides a collection of pre-trained models and a flexible pipeline for processing multimedia data, such as images, video streams, and audio.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



VIII. SYSTEM TESTING

Functional testing verifies that the system operates according to its defined specifications. It ensures that all features, including gesture recognition, text-to-speech conversion, and GUI interaction, perform correctly. This testing phase ensures that the system meets business and technical requirements and behaves as expected in real- world scenarios.

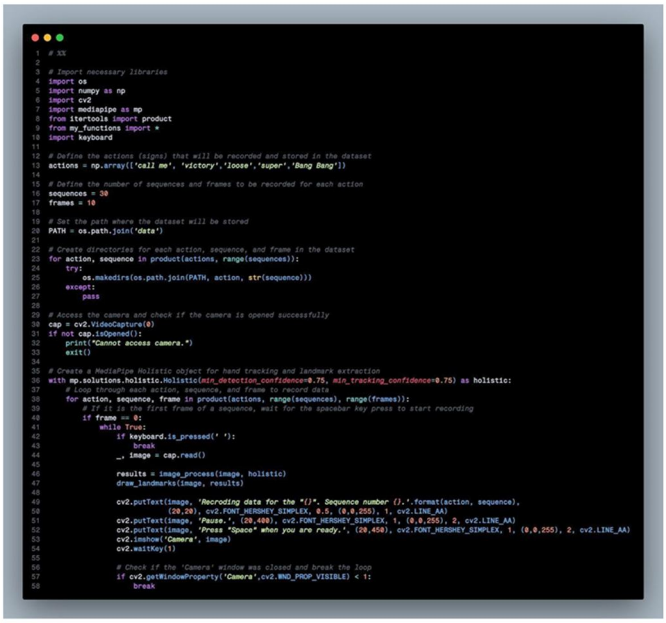
Unit testing involves testing individual components of the sign language detection system to ensure that they function correctly in isolation. It is essential for validating the internal logic of machine learning models, preprocessing steps, and GUI elements. This type of testing helps identify and fix errors early in the development phase.

Integration testing verifies that all software components work together as a unified system. Although individual modules may function correctly in isolation, they must also be tested in combination to identify any inconsistencies or failures. The goal is to ensure seamless data flow between different components and maintain stability under various scenarios.

IX. IMPLEMENTATION

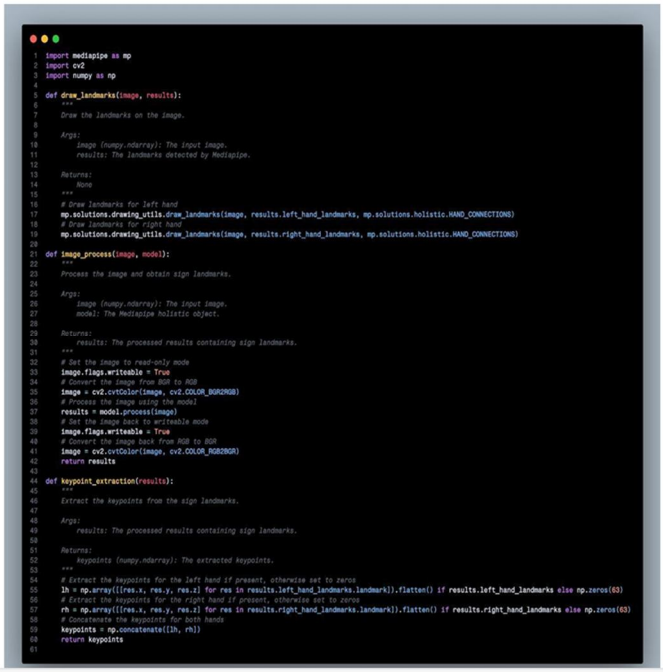
A. Source Code

1) Module 1: collection.py



```
# 3X

# Import necessary libraries
import os
import numpy as np
import cv2
import mediapipe as mp
from itertools import product
from my_functions import *
import keyboard

# Define the actions (signs) that will be recorded and stored in the dataset
actions = np.array(['call me', 'victory','loose','super','Bang Bang'])

# Define the number of sequences and frames to be recorded for each action
sequences = 30
frames = 10

# Set the path where the dataset will be stored
PATH = os.path.join('data')

# Create directories for each action, sequence, and frame in the dataset
for action, sequence in product(actions, range(sequences)):
    try:
        os.makedirs(os.path.join(PATH, action, str(sequence)))
    except:
        pass

# Access the camera and check if the camera is opened successfully
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("Cannot access camera.")
    exit()

# Create a MediaPipe Holistic object for hand tracking and landmark extraction
with mp.solutions.holistic.Holistic(min_detection_confidence=0.75, min_tracking_confidence=0.75) as holistic:
    # Loop through each action, sequence, and frame to record data
    for action, sequence, frame in product(actions, range(sequences), range(frames)):
        # If it is the first frame of a sequence, wait for the spacebar key press to start recording
        if frame == 0:
            while True:
                if keyboard.is_pressed(' '):
                    break
                _, image = cap.read()

                results = image_process(image, holistic)
                draw_landmarks(image, results)

                cv2.putText(image, 'Recroding data for the "{}", Sequence number {}.'.format(action, sequence),
                            (20,20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1, cv2.LINE_AA)
                cv2.putText(image, 'Pause.', (20,400), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2, cv2.LINE_AA)
                cv2.putText(image, 'Press "Space" when you are ready.', (20,450), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2, cv2.LINE_AA)
                cv2.imshow('Camera', image)
                cv2.waitKey(1)

                # Check if the 'Camera' window was closed and break the loop
                if cv2.getWindowProperty('Camera',cv2.WND_PROP_VISIBLE) < 1:
                    break
```

2) Module 2: my_functions.py



```python
import mediapipe as mp
import cv2
import numpy as np

def draw_landmarks(image, results):
    """
    Draw the landmarks on the image.

    Args:
        image (numpy.ndarray): The input image.
        results: The landmarks detected by Mediapipe.

    Returns:
        None
    """
    # Draw landmarks for left hand
    mp.solutions.drawing_utils.draw_landmarks(image, results.left_hand_landmarks, mp.solutions.holistic.HAND_CONNECTIONS)
    # Draw landmarks for right hand
    mp.solutions.drawing_utils.draw_landmarks(image, results.right_hand_landmarks, mp.solutions.holistic.HAND_CONNECTIONS)

def image_process(image, model):
    """
    Process the image and obtain sign landmarks.

    Args:
        image (numpy.ndarray): The input image.
        model: The Mediapipe holistic object.

    Returns:
        results: The processed results containing sign landmarks.
    """
    # Set the image to read-only mode
    image.flags.writeable = True
    # Convert the image from BGR to RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Process the image using the model
    results = model.process(image)
    # Set the image back to writeable mode
    image.flags.writeable = True
    # Convert the image back from RGB to BGR
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    return results

def keypoint_extraction(results):
    """
    Extract the keypoints from the sign landmarks.

    Args:
        results: The processed results containing sign landmarks.

    Returns:
        keypoints (numpy.ndarray): The extracted keypoints.
    """
    # Extract the keypoints for the left hand if present, otherwise set to zeros
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else np.zeros(63)
    # Extract the keypoints for the right hand if present, otherwise set to zeros
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else np.zeros(63)
    # Concatenate the keypoints for both hands
    keypoints = np.concatenate([lh, rh])
    return keypoints
```

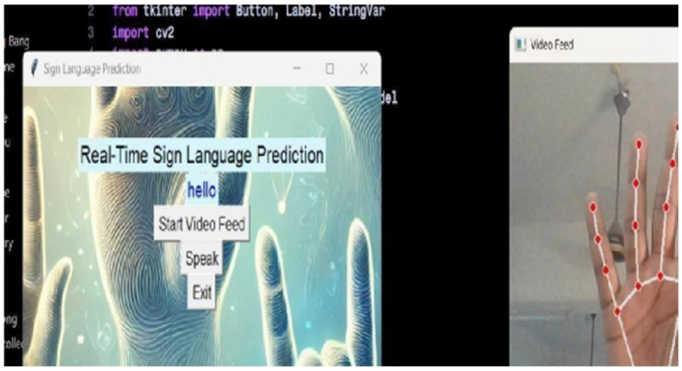
3) Module 3: model.py

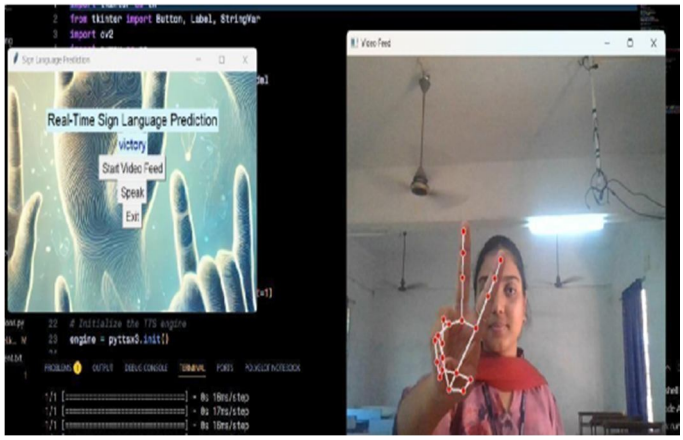
```python
# KK

# Import necessary libraries
import os
import numpy as np
import cv2
import mediapipe as mp
from itertools import product
from my_functions import *
import keyboard

# Define the actions (signs) that will be recorded and stored in the dataset
actions = np.array(['call me', 'victory','loose','super','Bang Bang'])

# Define the number of sequences and frames to be recorded for each action
sequences = 30
frames = 10

# Set the path where the dataset will be stored
PATH = os.path.join('data')

# Create directories for each action, sequence, and frame in the dataset
for action, sequence in product(actions, range(sequences)):
    try:
        os.makedirs(os.path.join(PATH, action, str(sequence)))
    except:
        pass

# Access the camera and check if the camera is opened successfully
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("Cannot access camera.")
    exit()

# Create a MediaPipe Holistic object for hand tracking and landmark extraction
with mp.solutions.holistic.Holistic(min_detection_confidence=0.75, min_tracking_confidence=0.75) as holistic:
    # Loop through each action, sequence, and frame to record data
    for action, sequence, frame in product(actions, range(sequences), range(frames)):
        # If it is the first frame of a sequence, wait for the spacebar key press to start recording
        if frame == 0:
            while True:
                if keyboard.is_pressed(' '):
                    break
                _, image = cap.read()

                results = image_process(image, holistic)
                draw_landmarks(image, results)

                cv2.putText(image, 'Recroding data for the *{}*, Sequence number {},'.format(action, sequence),
                            (20,20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1, cv2.LINE_AA)
                cv2.putText(image, 'Pause.', (20,400), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2, cv2.LINE_AA)
                cv2.putText(image, 'Press "Space" when you are ready.', (20,450), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2, cv2.LINE_AA)
                cv2.imshow('Camera', image)
                cv2.waitKey(1)

                # Check if the 'Camera' window was closed and break the loop
                if cv2.getWindowProperty('Camera',cv2.WND_PROP_VISIBLE) < 1:
                    break
        else:
            # For subsequent frames, directly read the image from the camera
            _, image = cap.read()
            # Process the image and extract hand landmarks using the MediaPipe Holistic pipeline
            results = image_process(image, holistic)
            # Draw the hand landmarks on the image
            draw_landmarks(image, results)

            # Display text on the image indicating the action and sequence number being recorded
            cv2.putText(image, 'Recroding data for the *{}*, Sequence number {},'.format(action, sequence),
                        (20,20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1, cv2.LINE_AA)
            cv2.imshow('Camera', image)
            cv2.waitKey(1)

        # Check if the 'Camera' window was closed and break the loop
        if cv2.getWindowProperty('Camera',cv2.WND_PROP_VISIBLE) < 1:
            break

        # Extract the landmarks from both hands and save them in arrays
        keypoints = keypoint_extraction(results)
        frame_path = os.path.join(PATH, action, str(sequence), str(frame))
        np.save(frame_path, keypoints)

    # Release the camera and close any remaining windows
    cap.release()
    cv2.destroyAllWindows()
```

©IJRASET: All Rights are Reserved

4) Module 4: run.py

X. OUTPUT SCREENSHOTS





XI. CONCLUSION

The Sign Language Detection System developed in this project integrates computer vision, deep learning, and real-time processing to bridge the communication gap for the deaf and hard-of-hearing community. The system utilizes CNN-LSTM models for dynamic gesture recognition, MediaPipe and OpenCV for accurate hand tracking, and a Tkinter-based GUI for seamless interaction. The integration of real-time video processing and AI-driven sign recognition ensures that users can communicate effectively, with minimal latency and high accuracy. The system is designed to be scalable, adaptable, and efficient for various sign language dialects. The technical feasibility of the project demonstrates that the system supports multi-language recognition, real-time processing, and cloud or edge-based execution. The use of transfer learning and data augmentation techniques enables the model to expand its vocabulary without requiring an extensive dataset. The cost-benefit analysis suggests that this project could be implemented as an open-source tool or a commercially viable assistive technology for various sectors, including education, healthcare, and business environments.

XII. FUTURE SCOPE

The future scope of the sign language detection system extends to multilingual support, advanced AI-driven gesture recognition, integration with smart devices, and real-time accessibility enhancements. As deep learning evolves, the integration of transformer-based architectures, CNN-LSTM hybrids, and reinforcement learning will improve the system's ability to interpret complex and dynamic sign language sequences with higher accuracy and efficiency. Additionally, AI-powered speech-to-sign and sign to-speech conversion can enable seamless real-time communication between signers and non-signers, eliminating the need for human interpreters in various settings. The adoption of wearable technology, such as smart gloves, AR/VR headsets, and smart glasses, will further enhance gesture tracking and real-time translation, making the system more interactive and responsive. Cloud-based AI models will also support continuous learning and automatic updates, enabling real-time improvements in gesture recognition.

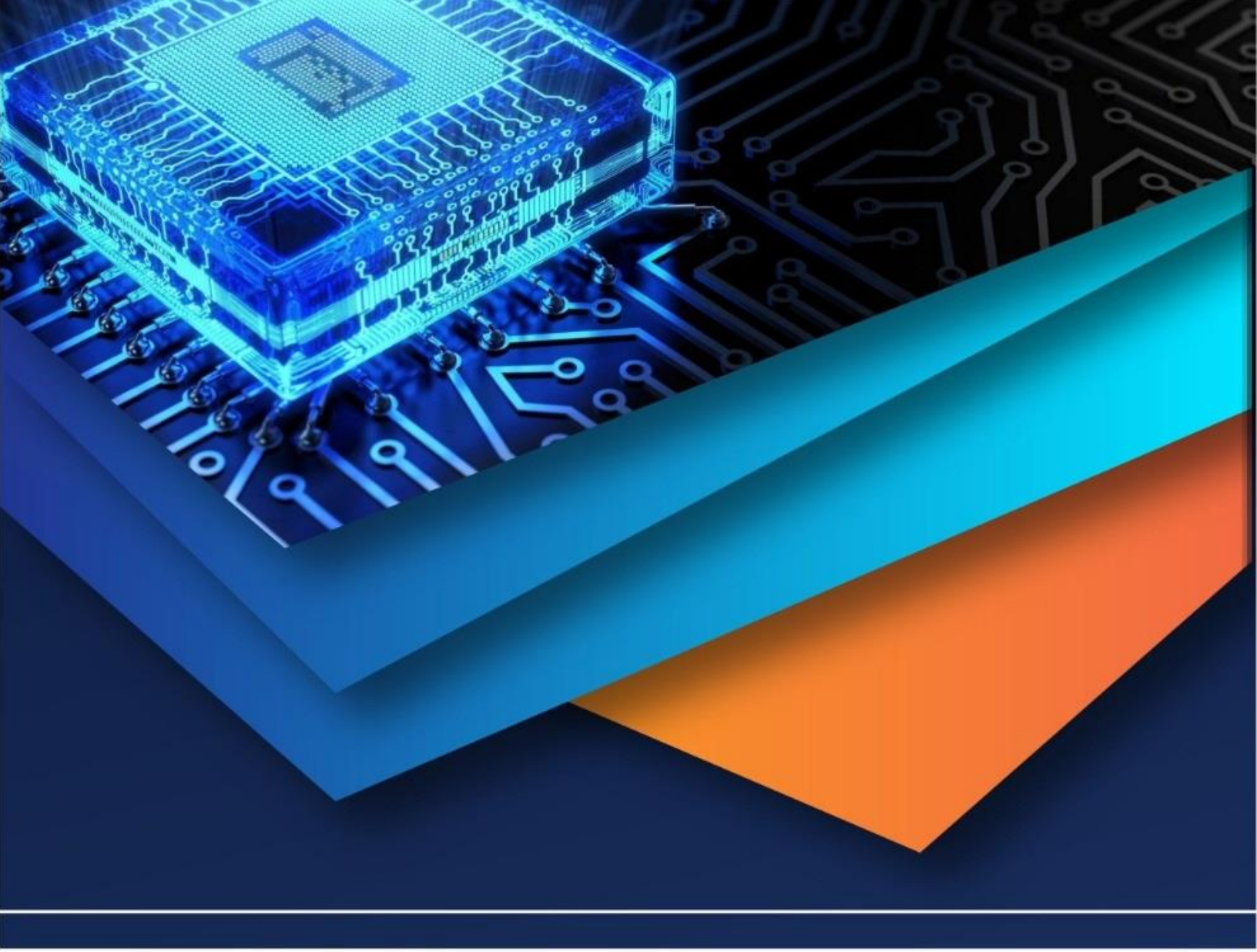
REFERENCES

[1] K. Cheng, "Top 10 & 25 American sign language signs for beginners – the most know top 10 & 25 ASL signs to learn first: Start ASL," Start ASL Learn American Sign Language with our Complete 3-Level Course, 29-Sep-2021. Top 10 & 25 American Sign Language Signs for Beginners – The Most Know Top 10 & 25 ASL Signs to Learn First.
[2] A. Mittal, P. Kumar, P. P. Roy, R. Balasubramanian, and B.B. Chaudhuri, "A Modified LSTM Model for Continuous Sign Language Recognition Using Leap Motion," in IEEE Sensors Journal, vol. 19, no. 16, pp. 7056-7063, 15 Aug.15, 2019.
[3] "Real time sign language detection with tensorflow object detection and Python | Deep Learning SSD," YouTube, 05-Nov-2020.
[4] V. Sharma, M. Jaiswal, A. Sharma, S. Saini and R. Tomar, "Dynamic Two Hand Gesture Recognition using CNN-LSTM based networks," 2021 IEEE International Symposium on Smart Electronic Systems (iSES), 2021, pp. 224- 229, doi: 10.11.09.
[5] K. Amrutha and P. Prabu, "ML Based Sign Language Recognition System," 2021 International Conference on Innovative Trends in Information Technology (ICITIIT), 2021, pp. 1-6, doi: 10.1109/ICITIIT51526.2021.9399594. Available: ML Based Sign Language Recognition System | IEEE Conference Publication.
[6] M. Wurangian, "American sign language alphabet recognition," Medium, 15 Mar-2021 [Online]. Available: American Sign Language Alphabet Recognition | by Marshall Wurangian | MLearning.ai | Medium.

Author's Profiles

[1]Mrs. J.MADHURI is currently working as a Assistant Professor in the Department of Computer Science and Engineering at PBR Visvodaya Institute of Technology and Science, Kavali, SPSR Nellore, Andhra Pradesh, India. She had 8 years of academic experience. She earned master of technology in Computer Science and Engineering. She had Published 5 research papers in various International Journals Research areas are Deep Learning and Machine Learning.
Email- Madhurijanvi@gmail.com

[2]Mrs. M.MOUNIKA, currently pursuing M.Tech in the Department of Computer Science and Engineering at PBR Visvodaya Institute of Technology and Science, Kavali, SPSR Nellore, Andhra Pradesh, India.
Email- mounika1291@gmail.com



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)