St. Xavier's College [Autonomous], Kolkata

Department of Computer Science

30, Mother Teresa Sarani, Kolkata-700016

VIRTUAL ASSISTANT FOR MUTE PEOPLE:

Detection of hand signs to perform various activities by

Aditya Vikram Sharma (517)
Ashish Lal (556)
Mounav Bhattacharjee (550)

M.SC. COMPUTER SCIENCE

SEMESTER IV

Under Guidance of

PROF. SHALABH AGARWAL

CERTIFICATE OF AUTHENTICATED WORK

This is to certify that the project report entitled “Virtual Assistant For Mute People: Detection of hand signs to perform various activities” submitted to the Department of Computer Science, St Xaviers College (Autonomous), in partial fulfilment of the requirement for the award of the degree of M.Sc. Computer Science is an entirely original work carried out by **Aditya Vikram Sharma** (Roll Number: 517), **Ashish Lal** (Roll Number: 556) and **Mounav Bhattacharjee** (Roll Number: 550) under the guidance of Prof. Shalabh Agarwal. The content embodied in this project is authentic and is genuine work done by the aforementioned students and has not been submitted whether to this college or to any other institute for the fulfilment of the requirement of any course of study.

Name and signature of the Project Team Members:

1. Aditya Vikram Sharma

2. Ashish Lal

3. Mounav Bhattacharjee

Signature of Guide:

Date: 28/04/2024

We hereby recommend that the project prepared above be accepted in partial fulfilment of the requirements of the degree of M.Sc. in Computer Science at St. Xaviers College (Autonomous), Kolkata.

Signature of Head of Department:

Date:

Signature of External Examiner:

Date:

ABSTRACT

In today's digital age, a virtual assistant has revolutionized lifestyle by enhancing productivity, accessibility, and convenience across various domains as it is designed to perform tasks and provide information or assistance to users often utilizing Artificial Intelligence and Natural Language Processing technologies to understand and respond to user queries and commands. It is a software application or program that provides services to users typically via voice commands or text-based interactions, individuals with speech impairments face significant challenges in expressing themselves effectively making these digital assistants largely ineffective for them. As technology continues to advance, virtual assistants are becoming increasingly sophisticated and capable of handling more complex tasks, therefore no one should be disenfranchised or deprived of the privilege of using them. Hence, we want to propose a solution by developing a novel system tailored specifically for mute users to serve as a real-time sign language translator and can potentially be integrated into devices like virtual assistants.

ACKNOWLEDGEMENT

This project, where we developed a system that communicates with users utilizing hand gestures and assists them by providing desired services was extremely beneficial in terms of the overall knowledge gained. We would like to express our sincere gratitude to all those who contributed to the completion of this project. However, the project would not have been possible without the help of a select few. First and foremost, we extend our appreciation to Prof. Shalabh Agarwal, whose guidance and expertise were invaluable throughout this endeavour. Their insights and support significantly enhanced the quality and depth of our work. We are also deeply thankful to faculty members of the Department of Computer Science, St. Xavier's College (Autonomous), Kolkata for their unwavering commitment and dedication. Their tireless efforts and attention to detail played a crucial role in achieving our objectives. Additionally, for providing us with the opportunity to undertake this vast and rich project we extend our gratitude to St. Xavier's College (Autonomous) renowned for its commitment to academic excellence, holistic education, and fostering a vibrant learning community. Finally, we are grateful to all our peers and batchmates who supported us throughout this journey their encouragement and cooperation were instrumental in overcoming challenges and reaching our goals. This project would not have been possible without the collective efforts of everyone involved, and for that, we are truly thankful.

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

Our project is grounded in machine learning, a subset of Artificial Intelligence (AI) that re lies on data-driven techniques to empower computers to acquire knowledge without explicit programming. Virtual assistant devices, exemplified by products like Amazon Echo or Google Home Mini, are meticulously engineered for integration within household or workplace settings, offering a diverse array of functionalities. These devices adeptly handle tasks ranging from playing music and providing informational responses to managing schedules and setting alarms. Their operation typically necessitates the utterance of predefined "wake words" to initiate device responsiveness, followed by verbal commands to execute desired tasks. Regrettably, digital assistants such as Siri or Amazon's Alexa exhibit significant limitations within the mute community due to their heavy reliance on auditory input and feedback from users. This reliance renders such assistants largely ineffective for individuals unable to articulate spoken commands, thereby presenting a considerable barrier to their accessibility and utility within this demographic. The World Health Organization stated that approximately 70 million people in the world are deaf-mutes. A total of 360 million people are deaf, and 32 million of these individuals are children.

As per the statistics provided by the World Health Organization, an estimated 70 million individuals worldwide are categorized as deaf-mute. Furthermore, there are approximately 360 million individuals who are classified as deaf, with 32 million of these belonging to the paediatric demographic.

1.1 BACKGROUND

Hand gesture recognition represents a natural mode of interaction between humans and computers and is a highly active field of study within computer vision and machine learning. The primary objective of research in gesture recognition for Human-Compute Interaction (HCI) is to design systems capable of identifying specific human gestures and utilizing them to convey information or control devices. To achieve this, vision-based hand gesture interfaces necessitate rapid and exceptionally robust hand detection and real-time gesture recognition. It's worth noting that sign languages are not standardized or universal; their grammatical structures vary from one country to another. Numerous prototypes can be readily expanded to encompass recognition of the entire English alphabet, thus serving as a solid foundation for the development of any vision-based sign language recognition user interface system.

1.2 OBJECTIVES

Our system is designed with the objective of enhancing communication accessibility for individuals with mutism. It endeavours to provide a means for such individuals to issue commands and request services through the utilization of hand gestures, thus circumventing

the reliance on verbal communication. It will employ a real-time model, utilizing a webcam to capture these hand signs, translating the visual input into standardized text, and subsequently relaying this information to the virtual assistant module. This innovative approach aims to empower mute individuals, enabling them to interact with technology in a remarkably efficient and effective manner.

1.3 PURPOSE, SCOPE, AND APPLICABILITY

PURPOSE

Our project is dedicated to the development of a specialized virtual assistant tailored explicitly for individuals afflicted with mutism. Our innovative virtual assistant provides a unique communication avenue, allowing users to express themselves through hand sign language. Employing a real-time model, our system utilizes webcam technology to capture hand signs, translating visual cues into standard text format, subsequently interfacing with the virtual assistant module. This pioneering approach is poised to empower mute individuals, facilitating efficient interaction with technology, and thereby augmenting their communication capabilities and overall quality of life.

The virtual assistant, adept in a multitude of functions, is designed to streamline daily activities such as scheduling reminders, placing calls, composing messages, and playing music, thereby fostering independence among mute users. Furthermore, it extends its utility to educational endeavours, offering assistance with learning materials, answering queries, and aiding in coursework completion. Our endeavour transcends mere facilitation of communication; it is rooted in the ethos of promoting autonomy, social inclusion, and holistic well-being. By harnessing the potential of technology, our project aspires to dismantle barriers and elevate the quality of life for individuals grappling with communication challenges.

SCOPE OF SYSTEM

1. Capturing and Tracking hand signs- facilitate real-time hand recognition through webcam input. The Mediapipe library is used to identify and segregate distinct hand landmarks. The system also tracks and monitors the detected hands in real-time. This tracking functionality enhances the system's capacity to discern hand signs effectively.
2. Sign Language Recognition- the virtual assistant will be designed to recognize and interpret various sign languages, allowing mute individuals who use sign language as their primary means of communication to interact with technology.
3. Real-time Translation- The virtual assistant would offer real-time translation services, converting sign language into spoken or written language and vice versa. This will be particularly beneficial in diverse linguistic environments.

4. Command Recognition and Execution- The refined textual data is translated into actionable commands. The system executes the commands to perform specific tasks like playing music, retrieving information, etc.
5. Assistance in Daily Activities- The virtual assistant can assist with daily tasks based on sign language commands, providing a hands-free way for individuals to control smart home devices, set reminders, make calls, send messages, play music, or perform other activities.
6. Accessibility in Public Spaces- Implementing the virtual assistant in public spaces, such as transportation hubs, offices, and hospitals, could enhance accessibility for mute individuals who use sign language. 8. Inclusion in Virtual Meetings- The virtual assistant can be integrated into virtual meeting platforms, enabling mute individuals who use sign language to participate in online discussions and collaborate effectively.

APPLICABILITY

Our system caters to individuals with mutism, offering them a means to communicate effectively using hand sign language. It enhances accessibility to technology and communication platforms, empowering users to engage in daily interactions, access services, and participate in social activities.

2. SURVEY OF TECHNOLOGIES

In the paper by Someshwar, Dipanshu, et al. "Implementation of virtual assistant with sign language using deep learning and TensorFlow." 2020 second international conference on inventive research in computing applications (ICIRCA). IEEE, 2020 work was done to implement a system using deep learning and TensorFlow that enables deaf individuals to interact with voice-controlled virtual assistants through sign language interpretation, bridging the accessibility gap for people with hearing and speaking disabilities. This system had a few limitations such as the need for a plain background and less accurate in terms of predicting gestures.

Similar work was carried out in the paper, T. J. Swamy, M. Nandini, N. B, V. Karthika K, V. L. Anvitha and C. Sunitha, "Voice and Gesture based Virtual Desktop Assistant for Physically Challenged People," 2022 6th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2022. This research work attempts to propose a voice and gesture-based virtual assistant that can be used by disabled as well as non-disabled persons to perform common tasks on their computers.

Another paper by Qi, Jing, et al. "Computer vision-based hand gesture recognition for human-robot interaction: a review." Complex & Intelligent Systems 10.1 (2024): 1581-1606, aimed to contribute to the understanding and improvement of the interaction between humans and computers, with the ultimate goal of creating more effective and enjoyable computing experiences for users.

3. REQUIREMENTS AND ANALYSIS

3.1 PROBLEM DEFINITION

Digital assistants like Siri or Amazon's Alexa face notable limitations when it comes to serving the mute community. Their heavy reliance on auditory input and feedback renders them largely inaccessible to individuals who cannot articulate spoken commands. Consequently, these assistants pose a significant barrier to accessibility and utility within this demographic. To address this challenge, it is imperative to devise solutions that cater to the unique communication needs of individuals within the mute community. This necessitates the development of digital assistant systems that offer alternative modes of interaction beyond reliance on auditory input and feedback. We can divide our problem into discrete subproblems, thereby enabling a structured approach to system development. By breaking down the problem into manageable modules, we can focus on addressing each component individually, leading to more efficient development and greater flexibility in system design.

These modules encompass the following:

1. Hand Capture Module: This module is designed to facilitate real-time hand recognition through webcam input. The Mediapipe library identifies and segregates distinct hand landmarks.

2. Hand Tracking Module: Building upon the Hand Capture Module, this component tracks and monitors the detected hands in real-time. This tracking functionality enhances the system’s capacity to discern hand gestures effectively.

3. Hand Gesture Recognition Machine Learning Module: Within this module, the hand gestures captured by the system are processed and recognized through machine learning algorithms. These recognized gestures are subsequently translated into corresponding text.

4. Command Recognition Module: Following text conversion, this module translates the refined text into actionable commands, aligning them with the appropriate command module for execution.

5. Command Execution Module: The final module is responsible for executing the commands derived from the preceding modules. It acts on the recognized commands, al lowing the system to perform specific tasks effectively.

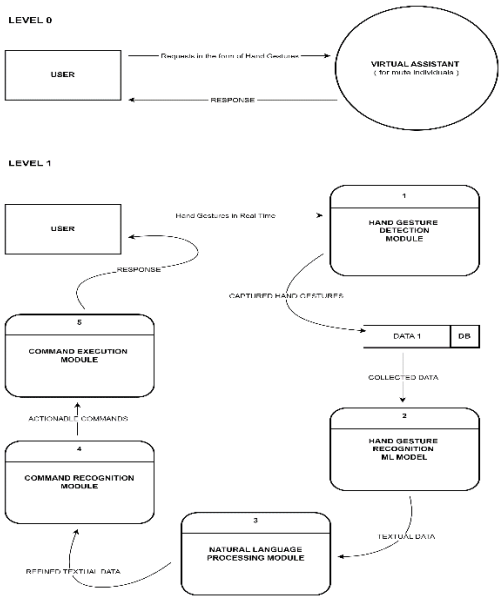

Fig 1: Level 0 and Level 1 Data Flow Diagram

3.2 REQUIREMENTS SPECIFICATION

FUNCTIONAL REQUIREMENTS:

Requirement ID: FR001

The virtual assistant will capture and recognize the hand signs of the user in the form of real-time video data. Acceptance Criteria:

1. The virtual assistant will have a built-in camera or utilize the device's camera to capture a live video feed of the user's hand gestures.

2. Distinct hand landmarks will be identified and segregated.

3. The system will initially support the American Hand Sign language. It will be further upgraded to support a wide range of hand sign Languages.

4. The virtual assistant will provide real-time feedback to the user, confirming the recognized hand sign and initiating the corresponding action or response.

Requirement ID: FR002

The virtual assistant will track and monitor the detected hands in real-time. This tracking functionality enhances the system’s capacity to discern hand gestures effectively. Acceptance Criteria:

1. This functionality is dependent on requirement id:001. The system employs hand-tracking algorithms for hand detection to identify and isolate the hands within the video feed.

2. The system will support tracking of both hands simultaneously to capture gestures involving both hands.

3. The virtual assistant shall be able to track hand movements adapting to varying distances and angles.

4. The hand-tracking functionality will be robust in different lighting conditions and environmental settings to ensure consistent performance.

5. The hand-tracking functionality will have a latency of no more than 100 milliseconds to provide a responsive and natural user experience.

6. The virtual assistant will include a calibration feature to optimize hand-tracking accuracy for individual users.

Requirement ID: FR003

The virtual assistant will make use of machine learning algorithms to accurately recognize and interpret hand signs used in sign language. Acceptance Criteria:

1. The captured hand frames are sent as input to a pre-trained machine-learning model capable of recognizing a predefined set of hand signs commonly used in sign language.

2. The machine learning model interprets the hand signs and translates them into textual representations.

3. The system will support continuous learning, allowing the machine learning model to adapt and improve its recognition accuracy over time based on user feedback and usage patterns.

4. The model will be trained on a dataset that includes variations in hand shapes, sizes, and orientations to ensure robust recognition across diverse user populations.

Requirement ID: FR004

The virtual assistant shall be capable of recognizing and executing user commands, allowing users to perform specific actions or tasks through signed instructions. Acceptance Criteria:

1. The virtual assistant translates the refined text from the NLP module into actionable commands which are then executed.

2. The virtual assistant will recognize a predefined set of command phrases or gestures that correspond to specific actions or tasks.

3. Commands will cover a range of functionalities, including but not limited to sending messages, setting reminders, initiating calls, and retrieving information.

4. The system will provide users with a command reference or list of available actions to facilitate user understanding.

5. The virtual assistant will confirm the execution of a command with a user-friendly response or feedback, providing a clear indication of the performed action.

6. The system will include error-handling mechanisms to address cases where a command is not recognized or if there are issues in executing the command.

NON-FUNCTIONAL REQUIREMENTS:

1. Usability:

● The virtual assistant's user interface will be designed to be intuitive and user-friendly, adhering to accessibility standards to accommodate users with varying degrees of motor skills.

● The system will provide alternative methods of interaction, such as touch or gesture controls, to cater to users who may have limitations in hand movements.

● The user interface of the virtual assistant will be designed with a clean layout ensuring ease of navigation.

● The system will allow users to customize preferences, including language settings, colour schemes, and interaction modes, to cater to individual user needs and preferences.

● The system shall provide a user-friendly onboarding process to introduce new users to the virtual assistant's features and functionalities.

● The virtual assistant shall offer contextual help and guidance, providing assistance or clarification when users encounter unfamiliar commands or situations.

2. Performance:

● The virtual assistant will respond to user inputs within seconds for both recognition and execution of commands to ensure real-time interaction.

● The system will optimize resource utilization, ensuring efficient use of CPU, memory, and network bandwidth to minimize the impact on device performance.

● The virtual assistant will have low network latency for sending and receiving data, to provide a responsive user experience.

● The virtual assistant will have limited offline functionality, allowing users to perform basic tasks even when not connected to the internet, with data synchronization upon reconnection.

● The virtual assistant will promptly handle errors, providing feedback to users within seconds of encountering an issue to maintain a smooth and transparent user experience.

3. Security:

● User data and personal information will be encrypted during transmission and storage to ensure privacy and comply with data protection regulations.

● The virtual assistant will implement secure authentication mechanisms to prevent unauthorized access to user data and system functionalities.

● User data, including voice recordings, hand sign patterns, and personal information, will be securely stored with encryption and access controls to prevent unauthorized access.

● The virtual assistant's development process shall follow secure coding practices, including regular security reviews, static code analysis, and dynamic application security testing (DAST).

● Software updates and patches will be delivered securely, with mechanisms in place to ensure the authenticity and integrity of updates, preventing tampering or malicious injection.

4. Reliability:

● The virtual assistant will be designed with fault-tolerant architecture, allowing the system to continue functioning seamlessly in the presence of hardware failures or other disruptions.

● Critical system components, such as databases, will have redundant backups to minimize the impact of hardware failures and enhance overall system reliability.

● Recovery procedures will be in place to restore system functionality quickly in the event of a failure, ensuring a reliable user experience.

● The virtual assistant will consistently and reliably recognize hand signs across different environments and lighting conditions, ensuring accuracy and minimizing recognition errors.

5. Adaptability

● The virtual assistant will be able to effectively track and recognize hand movements adapting to varying distances and angles.

● The system will be able to adapt to different lighting conditions and environmental settings to ensure consistent performance.

● The system will use adaptive language models to evolve its understanding of language nuances, dialects, and user-specific vocabulary, ensuring accurate interpretation of user queries.

● The virtual assistant shall utilize adaptable hand sign recognition models, allowing the system to learn and adapt to variations in users' signing styles over time.

3.3 SOFTWARE AND HARDWARE REQUIREMENTS

SOFTWARE REQUIREMENTS:

Python 3x-The code is implemented in Python, so we will need Python installed on our system.

Required Python Libraries to be installed-

1. **cv2 (OpenCV):**
    - Used for capturing video from the camera.
    - Enables various image processing tasks such as filtering, transformations, and object detection.

2. **mediapipe:**
    - Provides tools for hand tracking and landmarks detection.
    - Useful for recognizing hand gestures and movements in real-time.
3. **pickle:**
    - Allows for serialization and deserialization of Python objects.
    - Used here for loading and saving the trained machine learning model.
4. **numpy:**
    - Essential for numerical operations on multidimensional data arrays.
    - Provides efficient array operations and mathematical functions.
5. **webbrowser:**
    - Used to open web URLs, facilitating web-based interactions from the code.
6. **wikipedia:**
    - Enables fetching of information from Wikipedia.
    - Useful for retrieving summaries and details about various topics.
7. **pyttsx3:**
    - Facilitates text-to-speech conversion.
    - Used to convert text data into audible speech.
8. **pynput:**
    - Allows for controlling the keyboard and mouse inputs programmatically.
    - Useful for simulating user interactions and controlling the system.
9. **youtube_search:**
    - Provides functionalities for searching YouTube videos programmatically.
10. **pytube:**
    - Enables downloading of YouTube videos directly from the code.
    - Useful for fetching and saving videos for offline use or analysis.
11. **bs4 (Beautiful Soup):**
    - A web scraping library used for parsing HTML and XML documents.
    - Enables extraction of data from web pages.
12. **requests:**
    - Used for making HTTP requests to web servers.
    - Facilitates fetching of web content and data from URLs.
13. **scikit-learn:**
    - A comprehensive library for machine learning in Python.
    - Provides various algorithms, tools, and utilities for machine learning tasks.
14. **RandomForestClassifier:**
    - A machine learning algorithm for creating random forest classifier models.
    - Useful for classification tasks based on decision trees.
15. **train_test_split:**
    - A function for splitting datasets into training and testing sets.
    - Essential for model evaluation and validation.
16. **accuracy_score:**
    - A metric for evaluating the accuracy of machine learning models.
    - Measures the proportion of correctly classified instances.

17. confusion_matrix:

- Creates a confusion matrix to evaluate the performance of classification models.
- Provides insight into the model's predictive performance across different classes.

18. matplotlib.pyplot:

- A widely-used library for creating static, interactive, and animated plots.
- Allows for visualizing data in various forms such as line plots, histograms, and scatter plots.

19. seaborn:

- Built on top of matplotlib, seaborn offers enhanced data visualization capabilities.
- Provides attractive and informative statistical graphics.

20. GridSearchCV:

- GridSearchCV is imported from sklearn.model_selection module. It performs an exhaustive search over the parameter values that we have specified for our model

21. tensorflow.keras.preprocessing.sequence:

- A submodule of TensorFlow's Keras API for sequence preprocessing.
- Useful for tasks such as padding sequences, tokenization, and preparing data for neural networks.

To install all these libraries, we have to use pip installer and type the below code in the python terminal

“pip install opencv-python mediapipe numpy webbrowser wikipedia pyttsx3 pynput youtube-search-py pytube beautifulsoup4 requests scikit-learn matplotlib seaborn tensorflow”

HARDWARE REQUIREMENTS:

- **Webcam-**The code uses cv2.VideoCapture(0) to access the webcam. So, we need a webcam connected to your system.
- **Decent System Specifications-**The code involves real-time hand tracking and image processing, which can be computationally intensive. A modern multi-core processor (like Intel Core i5 or equivalent) is recommended for smooth performance. Having sufficient RAM (8GB or more) will ensure smooth execution.
- **Audio Output-**For the pyttsx3 library to work and read aloud the Wikipedia summaries, YouTube video names, and weather information, our system needs audio output capabilities. This could be speakers or headphones connected to our system.
- **Internet Connection-**The code accesses external resources such as Wikipedia, YouTube, and weather information from a website. So, an active internet connection is required for these functionalities to work.
- **Access Control-**We have to ensure that our system has the necessary permissions to access the webcam and make network requests.

4. SYSTEM DESIGN

4.1 CONCEPTUAL MODELS

The conceptual data model of a system outlines the comprehensive framework that encompasses data entities, attributes, relationships, and constraints. This model presents a high-level depiction of the system's data architecture, abstracting from intricate implementation details and considerations related to physical storage. Primarily, the conceptual data model functions as a fundamental blueprint, elucidating the data requisites and guiding the formulation of the system's data schema. It facilitates understanding and visualization of core data elements and their interconnections for stakeholders, including developers, designers, and end-users.

SOFTWARE MODEL

A software model is a conceptual depiction of a software system, facilitating comprehension, design, and documentation of its multifarious aspects. Acting as a foundational schematic or strategic outline, it orchestrates the construction process. Diverse in complexity and intent, software models find utility across various phases of the software development life cycle.

For our specific project, the Evolutionary Model approach was the preferred choice due to several compelling reasons:

1. Flexibility and Adaptability: The evolutionary model excels in situations where project requirements are not fully understood or are subject to change during development. It provides the necessary flexibility to adapt to evolving project needs.

2. Effective Response to Changes: This approach empowers the development team to respond to changing requirements in a nimble and efficient manner, which is crucial for ensuring project success.

3. Early Software Production: One of the key advantages of the evolutionary model is that it allows for the early production of software components. This early software production facilitates customer evaluation and feedback, which can guide the development process and improve the end-product.

4. Enhanced Risk Analysis: The evolutionary model enables better risk analysis and mitigation. By incrementally building and evaluating software, potential issues and challenges can be identified and addressed at an earlier stage, reducing project risks.

5. Support for Changing Environments: This approach is well-suited for projects operating in dynamic and changing environments, making it a robust choice for projects with evolving requirements and external factors.

6. Reduced Initial Operating Time: The evolutionary model can lead to a shorter initial operating time, as the project progresses incrementally, allowing for partial functionality to be available sooner.

7. Suitability for Large Mission-Critical Projects: It is particularly well-suited for large mission-critical projects, where the stakes are high, and a methodical and adaptable approach is essential for success.

By leveraging the strengths of the Evolutionary model, our project aims to address these crucial factors and optimize the development process for a successful outcome.

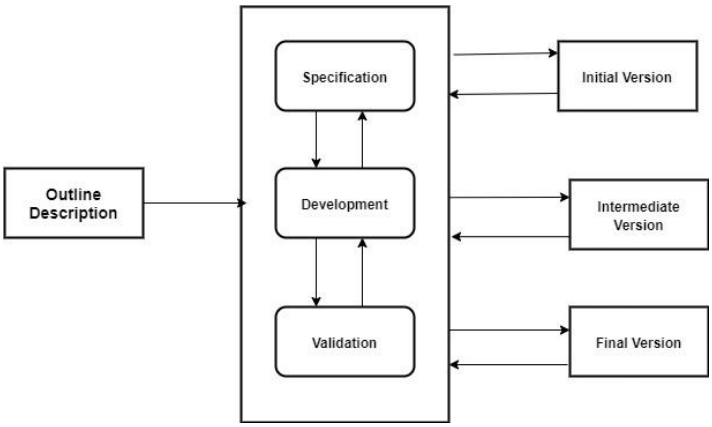

Fig 2: Evolutionary Software Process Model

Choosing an Evolutionary model approach for our project can be a strategic decision based on several reasons:

1. Adaptability: Evolutionary models, inspired by biological evolution, are well-suited for tasks where the system needs to adapt and improve over time. A virtual assistant can benefit from adapting to changing user needs and preferences.

2. Continuous Improvement: Evolutionary algorithms can continuously optimize the performance of the virtual assistant through iterations, allowing it to become more effective and user-friendly over time.

3. Complex Problem Solving: Virtual assistants often face complex and dynamic tasks, such as natural language understanding and generation. Evolutionary models can handle such complex problems by searching for better solutions incrementally.

4. Robustness: Evolutionary models can help create robust virtual assistants that can handle unexpected situations and recover from errors by evolving their strategies and behaviours.

5. Diversity: Evolutionary algorithms encourage diversity in the solution space, which can be beneficial for exploring a wide range of potential approaches and finding innovative solutions to challenges.

6. Limited Human Intervention: By using evolutionary models, you can reduce the need for manual programming and fine-tuning, making the development of the virtual assistant more autonomous.

7. Parallel Processing: Evolutionary algorithms can be parallelized to explore multiple solutions concurrently, potentially speeding up the development and learning process.

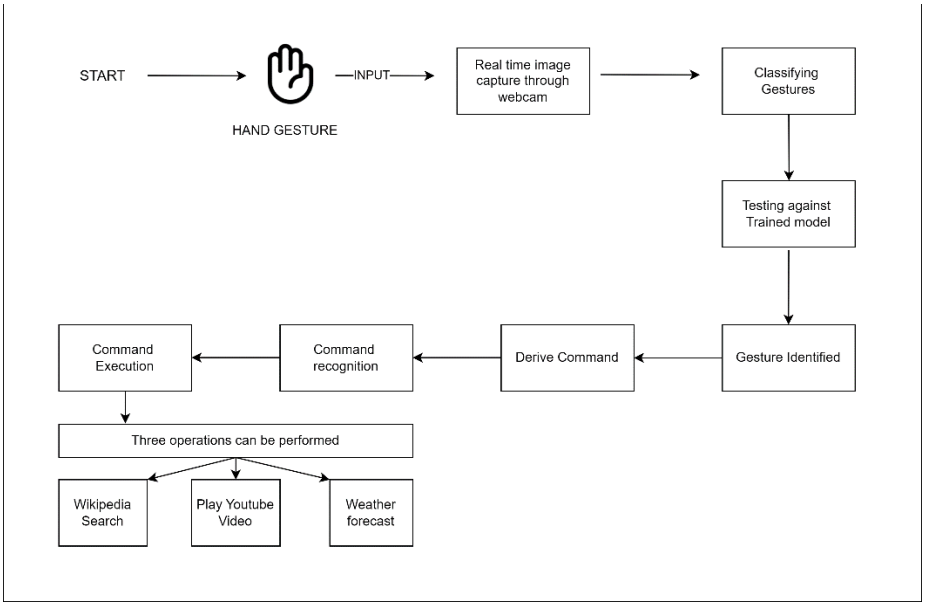

Fig 3: System Design Workflow

4.2 BASIC MODULES

Hand Gesture Detection Module:

- This module captures hand gestures in real-time using a webcam or camera-enabled device.
- Utilizes computer vision techniques to identify and track hand movements within the captured video feed.
- Outputs raw hand gesture data for further processing.

Hand Gesture Recognition Module:

- Processes raw hand gesture data from the detection module.
- Employs machine learning algorithms to extract features and classify hand gestures into predefined categories.
- Recognizes a wide range of hand gestures, including signs from sign language, gestures for commands, and navigational movements.

Conversion Module:

- Converts recognized hand gestures into standardized text format using speech-to-text conversion algorithms.
- Translates the interpreted gestures into textual representations of user commands or requests.

- Prepares the textual input for further processing by the command recognition module.

Command Recognition Module:

- Receives the textual input from the speech-to-text conversion module.
- Interprets the textual commands or requests, identifying the user's intent and desired actions.
- Utilizes natural language processing (NLP) techniques to understand the semantics and context of the input.

Command Execution Module:

- Executes the recognized commands or requests from the command recognition module.
- Performs tasks based on the interpreted commands, that include making Wikipedia search, playing videos on YouTube, and fetching weather forecasts.
- Interfaces with external systems and services to effectively fulfill user requests, ensuring seamless integration and reliable execution of tasks across multiple platforms and services.

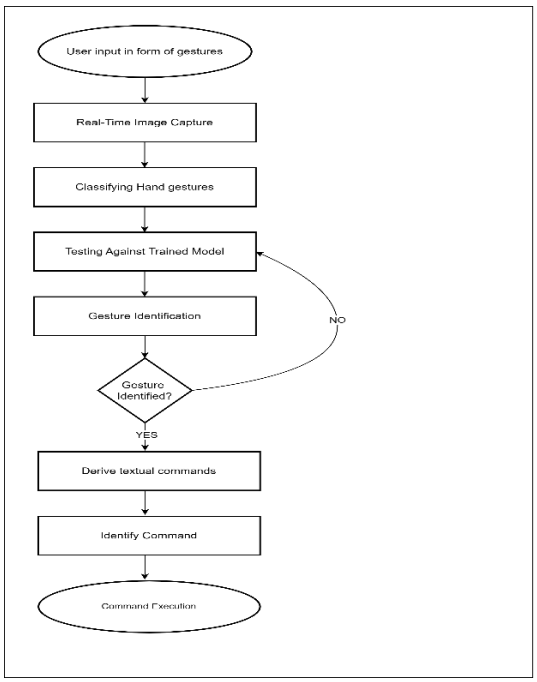

Fig 4: Process Flowchart

5. IMPLEMENTATION AND TESTING

5.1 DATASET DESCRIPTION

We have created a custom dataset for our project and it is based on the American Sign Language (ASL).
American Sign Language is a natural language that emerged from the American School for the Deaf (ASD) in Hartford during the 19th century. ASL has since then gained a lot of popularity and recognition.

Our dataset will have 26 classes and each class corresponds to a letter of the English alphabet. In American sign language for each of the 26 alphabets, there is a unique hand sign. So the dataset is created by collecting several images of hand signs for each alphabet/class.

5.2 DATA COLLECTION

We have used the built-in device camera to video capture and collect around 2000 images of our hands for each class of the data. We have used the OpenCV library in Python to perform this task. When we start the video capture process, a small window opens which is a waiting frame (waiting window). When we press a specific key, then it starts taking pictures of our hands. We can set a counter to keep track of the number of images captured. Once the counter reaches the required size, i.e., the required number of images have been captured for the first class, again the waiting frame (waiting window) appears and we need to again press the key to start capturing pictures for the next class.

We must be very careful while capturing the images and continuously move our hands slowly toward the camera and away from the camera so that the images of hand signs are captured at varying distances. Images should also be captured in different lighting conditions at different times of the day or in different locations so that our machine-learning model becomes robust.

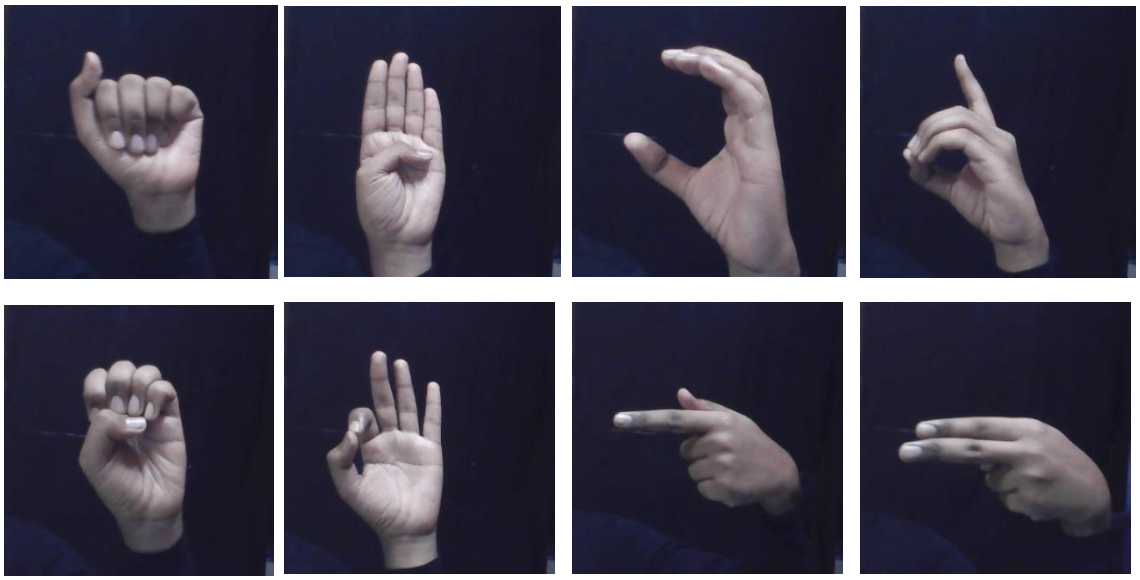

Fig 5: American Hand Sign Language Dataset

We take the help of the os library to create subdirectories for storing the images of each class. In a repetitive process, we capture images for each of the 26 classes, create data subdirectories labelled with the class numbers, and store the images in them. After this process is complete, we are left with a total of 26 subdirectories and 52000 images.

In the next step, we need to convert this image data into our dataset of suitable format.

5.3 DATA PREPROCESSING

Our objective is to perform classification on the different hand signs. But our collected images contain a lot of unwanted data such as the room background, the walls, our arms, shoulders, face, etc. All this data can be considered as noise since it has no role to play in the classification process. Also, we cannot work with raw image files for our machine-learning model. Our model requires numeric data for performing classification. So to remove all this noisy data, substantially reduce the dimensions of our dataset, and in turn perform vectorization, we take the approach of extracting Hand Landmarks from our images.

5.4 HAND LANDMARKS

Hand Landmarks indicate the key points of hands such as fingertips, knuckles, finger joints, wrist points, etc. We require the Media Pipe library, specifically the media pipe hands machine learning model for detecting and processing the hand landmarks in our images.

MediaPipe Hands is a high-quality hand and finger tracking solution. the hand landmark model uses machine learning to extract 21 3D landmarks of a hand using regression from just a single frame. It is a very robust model that works even for partially visible hands and handles self-occlusion.
Now there are certain configuration options such as:

- static_image_mode- By default, it is set to false. But we have set this parameter to true so that the hand detection runs on every input image. This is suitable in our case for processing a set of static images for each of the classes.

- min_detection_confidence- It denotes the minimum confidence value from the hand detection model for the detection to be considered as successful. The default value is 0.5. We have set this value to 0.3.

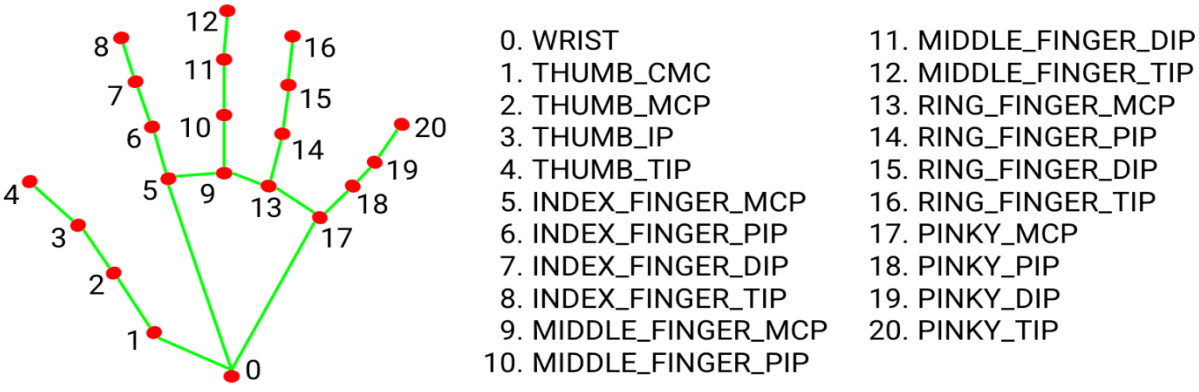

Fig 6: 21 3D Hand Landmarks

5.5 DATASET CREATION

We iterate through all the directories of images and for each image we process and store the hand landmarks. The media pipe hand landmarks model works best with RGB images. So after reading the images using opencv, we need to convert it from BGR to RGB format. Next, we extract the x and y coordinates of the 21 processed hand landmarks of each image and store them in an auxiliary array. Finally, we store all the auxiliary arrays into one big data array. The subdirectory names make up the respective class labels. makes up our dataset. The data array and the class labels together form our dataset.

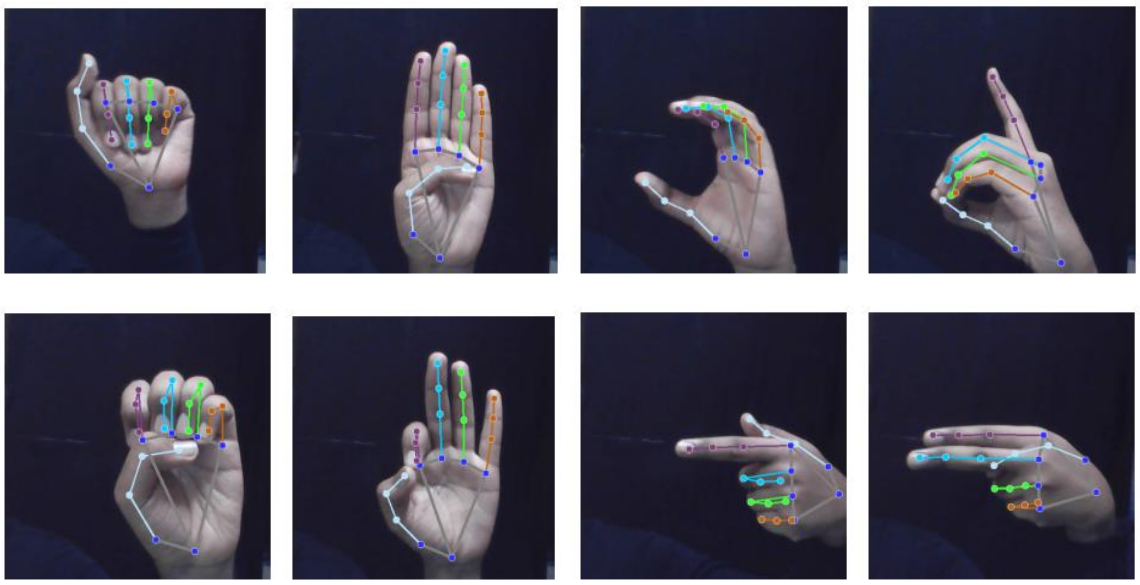

Fig 7: Drawing Hand Landmarks

5.6 DATASET PICKLING/UNPICKLING

We need to save our dataset by pickling it. Pickling is a process by which we can store our dataset in the form of a binary file. We do this because the process of creating the dataset by extracting the hand landmarks from the thousands of collected images is very time-consuming. To avoid repeating this process each time the device is powered on, we save the dataset as a pickle file. A pickle file is a binary file that allows us to quickly load and get back the original dataset thus saving a lot of time.

5.7 MULTICLASS CLASSIFICATION

A classification problem involving more than 2 classes is known as Multiclass classification. Our objective is to predict and assign each hand sign to one of the 26 classes without overlapping.

Some popular multiclass classifiers include K Nearest Neighbors (KNN), Naive Bayes, Decision Trees, and Random Forest Classifiers. Each classifier has its restrictions but there is no such rule that implies a particular classifier is suitable for a particular dataset. Therefore to determine which classifier is best suited for the task, we apply various classifiers to our dataset and calculate the various performance metrics. The classifier with the best performance metric is then chosen.

5.8 CLASSIFIER PERFORMANCE METRICS

The classification models give us predicted outputs which can then be compared with the actual outputs(ground truth) to get an idea of the predictive capabilities. Several metrics are available to determine the performance of the classifier. Choosing the suitable classifier performance metric is crucial in evaluating different models and approaches.

Some important metrics:

1. Confusion Matrix- This is a matrix displaying the True Positives, True Negatives, False Positives, and False Negatives predicted by the classifier.

2. Accuracy/Recognition Rate- It determines how accurately the model is predicting the outcome. However, it is not a good measure if the data samples are unbalanced.

3. Error/Misclassification Rate- This determines the rate at which the model is incorrectly classifying the data.

4. Precision- It is a measure of exactness.

5. Recall- It is a measure of completeness.

6. F1 score- This metric is the harmonic mean of precision and recall.

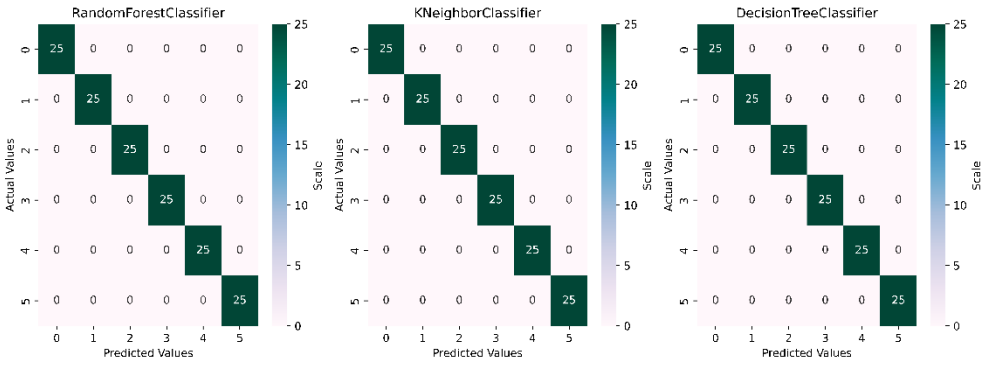

Fig 8: Comparative Confusion Matrix

5.9 RANDOM FOREST CLASSIFIER

We have decided to use a Random Forest Classifier for the multiclass classification problem. It works on the concept of ensemble learning, i.e., combining multiple classifiers to improve the accuracy of prediction of the model and to control over-fitting.
Instead of relying on a single decision tree, the Random Forest classifier takes the prediction output from several decision trees on subsets of the given dataset.
The higher the number of trees in the forest, the higher the prediction accuracy and it also prevents the problem of overfitting. The Decision Trees in the forest use the best-split strategy to divide into sub-nodes. Nodes that do not affect the result are removed by method of pruning.

5.10 RANDOM FOREST PARAMETERS

The classifier parameters that have been used are:

n_estimators- It denotes the number of decision trees in the forest. The default value is 100.

criterion *{"gini", "entropy", "log_loss"}*-This function is used to understand and measure the amount of unpredictability/impurity in the given dataset. The higher the value of entropy, the harder it is to get any conclusive result. Supported criteria are "gini" for the Gini impurity whereas "log_loss" and "entropy" both are for the information gain. Splitting is done based on that attribute which gives us the maximum information gain. The default value is "gini".

max_depth- Allows us to specify the maximum depth of the decision tree. If the value is None, then nodes are expanded until all leaves of the tree are pure or they contain samples less than min_samples_split. The default value is None.

min_samples_split- It denotes the minimum number of samples that are required to split an internal node. The default value is 2.

min_samples_leaf- It is the minimum number of samples required to be at the leaf node of a decision tree. This parameter has the effect of smoothing the model in case of regression. The default value is 1.
min_weight_fraction_leaf-
This parameter denotes the minimum weighted fraction of the total weights of input samples that are present at a leaf node. If weight is not provided then samples have equal weights. The default value is 0.0

max_features*{"sqrt", "auto", None}*-
The number of features that have to be considered while searching for the best split. With each split data becomes more and more homogeneous thus decreasing the impurity of the given dataset. The default value is "sqrt", i.e., the maximum features is sqrt(n_features).

5.11 HYPERPARAMETER TUNING

When we are working with a machine learning model, the model can be configured according to our requirements with the help of different Hyperparameters. Tuning of Hyperparameters refers to the process of optimizing the hyperparameters of the machine learning model. Tuning is performed so that our model may generalize well to unseen data i.e. reduce the problem of overfitting. It also significantly improves the performance of the model, such as accuracy, precision, recall, and in turn, producing the best results

We need to set the Hyperparameters before the training process begins. We try different combinations of hyperparameters and evaluate the performance of the model using a validation set or cross-validation. Here we have used the technique of Grid Search, which automates the process of hyperparameter tuning. To find the optimal set of hyperparameters we need to perform multiple experiments.

Here are the hyperparameters on which we have performed tuning:

1. n_estimators=10 numbers in the range [10,100]

2. max_features=['auto', 'sqrt']

3. max_depth=[2,4]

4. min_samples_split=[2,5]

5. min_samples_leaf=[1,2]

6. bootstrap=[True,False]

We create a parameter grid, which is a dictionary of the hyperparameter values.

```
In [7]: print('Parameter Grid:\n',param_grid)
Parameter Grid:
 {'n_estimators': [10, 17, 25, 33, 41, 48, 56, 64, 72, 80],
'max_features': ['auto', 'sqrt'], 'max_depth': [2, 4],
'min_samples_split': [2, 5], 'min_samples_leaf': [1, 2],
'bootstrap': [True, False]}
```

Fig 9: Hyper Parameter Grid

The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

GridSearchCV is imported from sklearn.model_selection module. It performs an exhaustive search over the parameter values that we have specified for our model. Function parameters taken are:

- **cv-** It is the cross-validation generator. It determines the cross-validation splitting strategy. We have taken cv=3, i.e., 3-fold cross-validation.

- **verbose-** This controls the verbosity. The higher the value, the more messages it displays. Value is taken as 2.

- **n_jobs-** Value controls the number of jobs to run in parallel. We have used n_jobs=4.

GridSearchCV implements a “fit” and a “score” method. After fitting the grid with the training dataset, the function performs statistical analysis to determine which set of hyperparameters gives the best result for our model. The training and the testing accuracy are then calculated. Ultimately, we save the best parameter model as a pickle file. Hyperparameter tuning is a very crucial but computationally intensive process.

5.12 COMMAND EXECUTION MODULE

The command execution module follows the machine learning module in sequence, taking the predicted character output from the machine learning process as its input. It then determines which operation to execute from a set of three options: searching for information on Wikipedia, playing a YouTube video using a web browser, or providing the current weather information for the location.

The three modules for commands are:

(A) **Wikipedia_info(formed_word):-** It fetches a summary of a given word from Wikipedia and reads it aloud using text-to-speech (TTS). Here's a breakdown of the code along with an algorithmic representation:

Algorithm-

1. **Setting Wikipedia Language:**
   wikipedia.set_lang("en"): Sets the language of the Wikipedia module to English.
2. **Fetching Wikipedia Summary:**
   summary = wikipedia.summary(formed_word, sentences=2): Retrieves a summary of the formed_word from Wikipedia, limited to two sentences. The summary is stored in the summary variable.
3. **Printing Wikipedia Summary:**
   print("Wikipedia Summary:", summary): Prints the fetched Wikipedia summary to the console.
4. **Text-to-Speech (TTS) Engine Initialization**:
   engine = pyttsx3.init(): Initializes a text-to-speech (TTS) engine using the pyttsx3 library.
5. **Setting Speech Rate:**

engine.setProperty('rate', 150): Sets the rate of speech for the TTS engine. The value 150 represents the speed of speech.

6. **Reading Aloud the Summary**:
   engine.say(summary): Instructs the TTS engine to read aloud the summary fetched from Wikipedia.
7. **Executing the TTS Engine:**
   engine.runAndWait(): Executes the TTS engine, which reads the summary aloud. The program will wait until the speech is completed before moving on.
8. **Exception Handling:**The code includes several except blocks to handle potential errors:
   - wikipedia.exceptions.PageError: Handles the case when the Wikipedia page for the given formed_word does not exist. It prints a message indicating that the Wikipedia page was not found.
   - wikipedia.exceptions.DisambiguationError: Deals with cases where the term provided is ambiguous and may refer to multiple Wikipedia pages. It prints a message asking for a more specific word.
   - Exception: Catches any other unexpected exceptions that might occur during the execution of the function. It prints the error message to the console.
9. **Finish**

(B).**youtube_video(formed_word):-**It searches for a YouTube video related to a given formed_word and plays it in a web browser.

Algorithm-

1. **Searching for YouTube Video:**
   results = YoutubeSearch(formed_word, max_results=1).to_dict(): This line uses the YoutubeSearch library to search for YouTube videos related to the formed_word. It limits the search to one result (max_results=1) and stores the results in the results variable.
2. **Checking if Results Exist:**if results::
   Checks if the results list is not empty, meaning at least one video was found.
3. **Creating Video URL:**
   video_url = "https://www.youtube.com" + results[0]['url_suffix']: Constructs the URL of the first video in the search results. It appends the video's URL suffix to the base YouTube URL.
   .
4. **Opening Video in Web Browser:**
   webbrowser.open(video_url): Opens the constructed video_url in the default web browser, effectively playing the YouTube video.
5. **Printing Output:**
   If a video is found, it prints "Playing video for: formed_word" to the console. If no video is found, it prints "No video found for: formed_word" to the console.
6. **Exception Handling:**
   The code includes a try-except block to catch any exceptions that might occur during the execution:

Exception: Catches any general exceptions that might occur during the process, such as network errors or invalid search queries. It prints the error message to the console.

7. **Finish.**

(C)**read_weather():-**It fetches the current weather information for a specific location (in this case, Kolkata) from a weather website. It then reads the weather information aloud using text-to-speech (TTS).

Algorithm-

1. **Fetching Weather Information:**
   - url = "https://www.weather.com/en-IN/weather/today/l/INXX0028:1:IN": This line sets the URL of the weather page for the desired location (Kolkata in this case).
   - response = requests.get(url): It sends a GET request to the specified URL and stores the response in the response variable.
   - soup = BeautifulSoup(response.content, "html.parser"): This line creates a BeautifulSoup object soup to parse the HTML content of the response.
   - weather_info = soup.find("div", class_="CurrentConditions--phraseValue--2xXSr").text.strip(): It finds the HTML element with the specified class ("CurrentConditions--phraseValue--2xXSr") that contains the current weather information. The text content of this element is extracted and stored in the weather_info variable.
2. **Text-to-Speech (TTS) Engine Initialization:**
   - engine = pyttsx3.init(): Initializes a text-to-speech (TTS) engine using the pyttsx3 library.
   - engine.setProperty('rate', 150): Sets the rate of speech for the TTS engine. The value 150 represents the speed of speech.
3. **Reading Aloud the Weather Information:**
   - engine.say(f"The current weather in Kolkata is {weather_info}"): This line constructs a sentence with the fetched weather_info and instructs the TTS engine to read it aloud.
   - engine.runAndWait(): Executes the TTS engine, which reads the weather information aloud. The program will wait until the speech is completed before moving on.
4. **Exception Handling:**
   - The code includes a try-except block to catch any exceptions that might occur during the execution:
   - Exception: Catches any general exceptions that might occur during the process, such as network errors or parsing errors. It prints the error message to the console.
5. **Finish**

To determine which operation to run this algorithm has been used:

Algorithm:

1. **Conditional Check(‘if len(predicted_letters>=3’)-**It checks if the length of the predicted_letters list is greater than or equal to 3. This condition ensures that we have at least three predicted letters available to process.
2. **Extracting Last Three Predicted Letters (last_three_letters = set(predicted_letters[-3:]))-** It takes the last three elements of the predicted_letters list and converts them into a set. This set will contain the last three predicted letters.
3. **Performing Actions Based on Gesture-** The code then checks each set of last three letters for specific gestures and performs corresponding actions:
   (a).Gesture ‘W’- If the gesture is 'W', indicating a specific pattern (e.g., 'MMM' gesture), the code does the following:
   - Joins the predicted letters from index 5 onward into a single word.
   - Calls the wikipedia_info(formed_word) function to fetch and read aloud the Wikipedia information for the formed word.
   - Clears the predicted_letters list for the next word.

   (b) **Gesture 'Y':**If the gesture is 'Y', indicating another specific pattern, the code does the following:

   - Joins the predicted letters from index 5 onward into a single word.
   - Calls the youtube_video(formed_word) function to play a YouTube video related to the formed word.
   - Clears the predicted_letters list for the next word.

   (c). **Gesture 'T':**If the gesture is 'T', indicating a different pattern, the code does the following:

   - Calls the read_weather() function to read aloud the current weather information.
   - Clears the predicted_letters list for the next word.
4. **Clearing Predicted Letters (predicted_letters = []):**At the end of each block, the predicted_ letters list is cleared to prepare for the next set of predicted letters.

6. TESTING

Testing the entire model involves a structured approach known as unit testing and integration testing.

6.1 UNIT TESTING

Unit testing involves breaking down the entire code into smaller, individual modules. Each module is tested independently to ensure that it functions correctly and produces the expected output. In the context of our model, this means testing each function separately to verify its functionality.

Purpose:

- Verify the correctness of individual functions or modules.
- Ensure that each function behaves as expected, given specific inputs.

Testing Process:

- For instance, we would test the wikipedia_info function separately by providing it with various input words.
- We would test the youtube_video function by checking if it can correctly open and play YouTube videos based on input words.
- Similarly, other functions such as read_weather and gesture recognition would undergo individual testing.

Expected Outcome:

- Each function should produce the desired output or behaviour.
- Errors, exceptions, and edge cases should be handled appropriately.

6.2 INTEGRATION TESTING

Once we have verified that each individual module functions correctly in isolation, we move on to integration testing. Integration testing involves combining these tested modules together to ensure that they work harmoniously as a whole system.

Purpose:

- Validate interactions between different modules.
- Confirm that modules communicate and exchange data correctly.

Testing Process:

- In our case, we would integrate modules like gesture recognition, Wikipedia fetching, YouTube video playback, and weather information reading.
- We simulate different gestures or input scenarios to see if the model responds as expected.

- For example, we can simulate a 'W' gesture to trigger Wikipedia fetching, a 'Y' gesture for YouTube video playback, and a 'T' gesture for weather information.

Expected Outcome:

- The integrated system should exhibit the intended behaviour.
- Gestures should trigger the corresponding actions (fetching Wikipedia info, playing YouTube videos, reading weather).

The system should handle transitions between different states smoothly.

6.3 TESTING WORKFLOW

Unit Testing:

Create separate test cases for each function.

Verify that functions produce correct outputs for various inputs.

Ensure proper error handling for exceptions and edge cases.

Integration Testing:

Combine modules to form the complete system.

Test the system's functionality with simulated gestures or input sequences.

Confirm that the system behaves as expected in response to different inputs and gestures.

Overall Goal:

The goal of this testing approach is to ensure the reliability, functionality, and accuracy of the entire hand gesture recognition system. By thoroughly testing individual modules and their integration, we can identify and address any issues or bugs, resulting in a robust and user-friendly system.

Testing ‘wikipedia_info(formed_word):-

1. **Prepare Test Data:**
   - Define a few words or topics for which you want to retrieve Wikipedia summaries.
2. **Call the Function with Test Data:**
   - In our testing script or Python environment, import the necessary libraries (wikipedia and pyttsx3) .
   - Then, call the wikipedia_info function with our test words.

3. **Run the script in your terminal or preferred Python environment.**
   - This script will call the wikipedia_info function with the test words "Python", "Artificial Intelligence", and "Machine Learning".
   - The function will retrieve the Wikipedia summaries for these words and print them.
   - It will also use text-to-speech to read aloud each summary.

4. **Expected Output:**
   - For each test word, you should see the Wikipedia summary printed in the terminal.
   - The summaries will also be read aloud using the text-to-speech engine.

5. **Error Handling:**
   - Test the error handling by providing words that may not have Wikipedia pages.
   - For example, you can try providing a random string or a word that is not a common topic.
   - The function should gracefully handle errors such as Page Error and Disambiguation Error, printing appropriate messages.

Testing 'youtube_video(formed_word)'-

1. **Mock YoutubeSearch Results:**
   - Since the YoutubeSearch library is used to fetch YouTube search results, we need to mock its behaviour for testing purposes. We can create a mock object that simulates the behaviour of YoutubeSearch.

2. **Provide Test Cases:**
   - Create test cases that cover different scenarios:
     a. Scenario 1: Video found for the provided search term.
     b. Scenario 2: No video found for the provided search term.
     c. Scenario 3: Error occurs during the search process.

3. **Execute the youtube_video Function:**
   - Call the youtube_video function with different input parameters (formed words) in each test case.

4. **Expected Behavior:**
   - Verify the behaviour and output of the youtube_video function for each test case.
   - Use assertions to check if the function behaves as expected in different scenarios.

Testing 'read_weather()'

1. **Mock HTTP Response:**
   - Since the requests library is used to fetch weather information from a URL, we need to mock its behaviour for testing. We can create a mock object that simulates the behavior of requests.get.

2. **Provide Test Cases:**
   - Create test cases that cover different scenarios:
     a. Scenario 1: Successful retrieval of weather information.
     b. Scenario 2: Error occurs during the HTTP request or parsing.

3. **Execute the read_weather Function:**
   - Call the read_weather function in each test case.

4. **Expected Behaviour:**
   - Verify the behavior and output of the read_weather function for each test case.

7. CONCLUSION

7.1 LIMITATIONS

1. **Dependency on Webcam Quality:**
   - The model's performance is directly impacted by the quality of the webcam used. A higher resolution webcam provides clearer images, leading to better hand gesture detection and prediction.
   - Explanation: Lower quality webcams may introduce noise and distortion into the captured images, potentially affecting the accuracy of the model's predictions. Users may need to invest in a higher quality webcam for optimal performance.

2. **Sensitivity to Lighting Conditions:**
   - The model's accuracy is influenced by the lighting conditions in the user's environment. Adequate lighting is essential for clear image capture and accurate gesture recognition.
   - Explanation: Poor lighting can result in shadows, glare, or uneven illumination, making it challenging for the model to distinguish hand gestures effectively. Users are advised to ensure well-lit environments for optimal performance.

3. **Limited to Basic Alphabet Gestures:**
   - The current model is designed to recognize and interpret basic alphabet gestures from American Sign Language (ASL). It may not accurately detect more complex or nuanced signs.
   - Explanation: While the model performs well for basic ASL alphabet gestures, it may struggle with recognizing gestures outside this scope. Users should be aware of its limitations when using the system for communication.

4. **System Intensiveness:**
   - The model's computational requirements demand a system with sufficient processing power and memory. It is recommended to have at least an Intel i5 CPU and 8GB of RAM for smooth operation.
   - Explanation: Due to the real-time hand tracking and image processing involved, the model can be system-intensive. Users may experience lag or slower performance on systems with lower specifications. Investing in a capable system ensures optimal functionality and responsiveness.

7.2 FUTURE SCOPE

1. **Hardware Implementation:**
   - Currently, our model operates at the software level. There is an opportunity to enhance its functionality by implementing it at the hardware level. This could involve utilizing devices such as a Raspberry Pi along with audio and camera modules to create a standalone, portable device.

- Example: A user-friendly, self-contained system that interprets hand gestures for various commands, useful for individuals with limited mobility or those who require hands-free interaction with technology.

2. **Emergency Services Integration:**
   - The system's capabilities can be expanded to include features like calling emergency services. This would provide a critical lifeline for individuals, particularly those with speech-disabilities, in case of urgent situations.
   - Example: A gesture or specific hand sign could trigger the device to automatically call emergency services, sending the user's location for immediate assistance.
3. **Sign Language Translation:**
   - Transforming the system into a sign language to speech converter opens doors to inclusive communication for speech-disabled individuals. This feature could interpret sign language gestures and convert them into spoken words.
   - Example: The device could have a learning mode where it recognizes and associates sign language gestures with corresponding spoken words, aiding both the user and those interacting with them.

7.3 CONCLUSION

In summary, our model represents an exciting opportunity to improve how users interact with technology, especially catering to a wide range of needs and abilities. By combining machine learning algorithms with various Python libraries, we have created a system that can identify and interpret American Sign Language (ASL) alphabet gestures, translating them into actionable commands.

SOURCE CODE

COLLECTING IMAGES

```
import os
import cv2

# first step is to collect images of hand
#and store it in folder say "data"

# if such directory does not exist then create one
DATA_DIR = './data'
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)

# Handsigns- A,B,C,D,E,F,.....Z
number_of_classes = 26
dataset_size = 300 #for each class

# select camera index. By default main camera index is 0, but may vary
cap = cv2.VideoCapture(0)
# cap is a videocapture object

#coordinates for creating rectangle
start=[20,80]
end=[300,380]
color=(255, 0, 127) #in BGR format
```

```

# creating subdirectory for each class
for j in range(number_of_classes):
    if not os.path.exists(os.path.join(DATA_DIR, str(j))):
        os.makedirs(os.path.join(DATA_DIR, str(j)))

    print('Collecting data for class {}'.format(j))

    while True:
        # waiting frame
        ret, frame = cap.read()
        # mirror the frame vertically
        frame=cv2.flip(frame,1)

        # show text on the frame
        cv2.putText(frame,
                    'Press "Q" to Start!!',
                    (100, 50), #text pos
                    cv2.FONT_HERSHEY_DUPLEX, #font
                    1.3, #fontscale
                    color, #rgb color
                    2, #thickness
                    cv2.LINE_AA)
```

```
47
48          # Display rectangle
49          frame = cv2.rectangle(frame, (start[0], start[1]), (end[0], end[1]), color, 5)
50
51          # Naming window
52          cv2.namedWindow('Window', cv2.WINDOW_NORMAL)
53
54          # Resizing window
55          cv2.resizeWindow('Window', 700, 700)
56
57          # Moving window
58          cv2.moveWindow('Window', 20, 20)
59
60          # Displaying the frame
61          cv2.imshow('Window', frame)
62
63          if cv2.waitKey(25) == ord('q'):
64              break
65
66      counter = 0
67      #stop if 300 imgs captured for each class
68      while counter < dataset_size:
69          # reading frame
70          ret, frame = cap.read()
71          # mirror the frame vertically
72          frame=cv2.flip(frame,1)
73
74          #cropping frame as preview
75          cut_frame=frame[start[1]:end[1],start[0]:end[0]]
76
77          # Naming window
78          cv2.namedWindow('Preview', cv2.WINDOW_NORMAL)
79
80          # Moving window
81          cv2.moveWindow('Preview', 750, 20)
82
83          cv2.imshow('Preview', cut_frame)
84
85          # Displaying rectangle
86          frame = cv2.rectangle(frame, (start[0], start[1]), (end[0], end[1]), color, 5)
87
88          cv2.imshow('Window', frame)
89
90          # img captured every 100 msecs
91          if cv2.waitKey(100) == ord('e'):
92              break
93          cv2.imwrite(os.path.join(DATA_DIR, str(j), '{}.jpg'.format(counter)), cut_frame)
94
95          counter += 1
96
97      cv2.destroyWindow('Preview')
98
99  cap.release()
100 cv2.destroyAllWindows()
```

CREATING DATASET

```python
import os
import mediapipe as mp
import cv2
import pickle


# different mediapipe objects for displaying/drawing handlandmarks
mp_hands=mp.solutions.hands

# model for detecting hand landmarks
hands=mp_hands.Hands(static_image_mode=True,min_detection_confidence=0.3)

DATA_DIR='./data'
# DATA_DIR is the main data directory, dir_ refers to the subdirectories/classes

data=[]
labels=[]
for dir_ in os.listdir(DATA_DIR):
    print('Working on dir:',dir_)

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

        img=cv2.imread(img_path)
        # cv2 reads image in BGR so convert to RGB
        img_rgb=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # results stores all the image hand landmarks
        results=hands.process(img_rgb)

        # now iterating through results, if multiple landmarks found then...
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:

                for i in range(len(hand_landmarks.landmark)):
                    # print(hand_landmarks.landmark[i])
                    x=hand_landmarks.landmark[i].x #x coordinate
                    y=hand_landmarks.landmark[i].y #y coordinate
                    data_aux.append(x)
                    data_aux.append(y)

            data.append(data_aux[:42])
            labels.append(dir_)


print('Finished creating dataset')
```

```python

# now lets save our dataset by pickling it
with open('data.pickle','wb') as f:
    pickle.dump({'data':data,'label':labels},f)
```

DISPLAYING HAND LANDMARKS

```python
import os
import mediapipe as mp
import cv2
import matplotlib.pyplot as plt



# different mediapipe objects for displaying/drawing handlandmarks
mp_hands=mp.solutions.hands
mp_drawing=mp.solutions.drawing_utils
mp_drawing_styles=mp.solutions.drawing_styles


# model for detecting hand landmarks
hands=mp_hands.Hands(static_image_mode=True,min_detection_confidence=0.3)

DATA_DIR='./data'
# DATA_DIR is the main data directory, dir_ refers to the 4 subdirectories/classes


for dir_ in os.listdir(DATA_DIR):
    print('Displaying for class:',dir_)
    img_name=os.listdir(os.path.join(DATA_DIR, dir_))[0]
    img_path=os.path.join(DATA_DIR,dir_,img_name)
    #print(img_path)
    img=cv2.imread(img_path)

    # cv2 reads image in BGR so convert to RGB
    img_rgb=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # results stores all the image hand landmarks
    results=hands.process(img_rgb)

    # now iterating through results, if multiple landmarks found then...
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:

        #Drawing hand landmarks on the image
            mp_drawing.draw_landmarks(
                    img_rgb, #image to draw
                    hand_landmarks, #model output
                    mp_hands.HAND_CONNECTIONS, #hand connections
                    mp_drawing_styles.get_default_hand_landmarks_style(),
                    mp_drawing_styles.get_default_hand_connections_style()
                    )

    plt.figure()
    plt.axis('off')
    plt.imshow(img_rgb)
```

TRAINING CLASSIFIER

```
import pickle

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

import numpy as np

# Unpickling the dataset
with open('newdata.pickle','rb') as f:
    data_dict=pickle.load(f)

'''
#Only Run if problem occurs
rows=[]
for i in range(len(data_dict['data'])):
    l=len(data_dict['data'][i])
    if l!=42:
        print(i,l)
        rows.append(i)

#print(rows)

for i in rows:
    data_dict['data'][i]=data_dict['data'][i][:42]
'''


X=np.asarray(data_dict['data'])
y=np.asarray(data_dict['label'])


# Now we will split into training and testing dataset
X_train,X_test, y_train,y_test= train_test_split(X,y,test_size=0.25,shuffle=True,stratify=y)

# RandomForestClassifier used to train our model
rfmodel=RandomForestClassifier()


rfmodel.fit(X_train,y_train)

# Checking accuracy of our model
score1=rfmodel.score(X_test, y_test)
print("Model Accuracy=",score1)


# Saving/Pickling our model
with open('newmodel.p','wb') as f:
    pickle.dump({'model':rfmodel},f)
f.close()
```

HYPERPARAMETER TUNING

```python
#Hyper Parameter Tuning To remove overfitting

import pickle

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

import numpy as np


with open('newdata.pickle','rb') as f:
    data_dict=pickle.load(f)


X=np.asarray(data_dict['data'])
y=np.asarray(data_dict['label'])

X_train,X_test, y_train,y_test= train_test_split(X,y,test_size=0.25,shuffle=True,stratify=y)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```python

#number of trees
n_estimators=[int(x) for x in np.linspace(start=10,stop=80,num=8)]

#number of features to consider at each split
max_features=['auto','sqrt']

#max levels in the tree
max_depth=[2,4]

#min number of samples required to split a node
min_samples_split=[2,5]

#min number of samples required at each leaf node
min_samples_leaf=[1,2]

#Creating param grid
param_grid={'n_estimators':n_estimators,
            'max_features':max_features,
            'max_depth':max_depth,
            'min_samples_split':min_samples_split,
            'min_samples_leaf':min_samples_leaf,
            }
```

```python

print('Parameter Grid:\n',param_grid)

rfmodel=RandomForestClassifier()

from sklearn.model_selection import GridSearchCV

rf_grid=GridSearchCV(estimator=rfmodel, param_grid=param_grid,cv=3,verbose=2,n_jobs=4)


rf_grid.fit(X_train,y_train)


print(rf_grid.best_params_)

print(f'Accuracy={rf_grid.score(X_test,y_test):.3f}')

best_rf = rf_grid.best_estimator_

with open('bestmodel.p','wb') as f:
    pickle.dump({'model':best_rf},f)
f.close()
```

TESTING HAND SIGN RECOGNITION

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

# Different mediapipe objects to show landmarks
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)


# Loading/Unpickling our model
with open('newmodel.p','rb') as f:
    model_dict=pickle.load(f)
f.close()

model=model_dict['model']


# Create a labels dictionary
labels_dict={0:'A',1:'B',2:'C',
             3:'D',4:'E',5:'F',
             6:'G',7:'H',8:'I',
             9:'J',10:'K',11:'L',
             12:'M',13:'N',14:'O',
             15:'P',16:'Q',17:'R',
             18:'S',19:'T',20:'U',
             21:'V',22:'W',23:'X',
             24:'Y',25:'Z'}

start=[20,80]
end=[300,380]
color=(255, 0, 127) #bgr format

cap=cv2.VideoCapture(0)

while True:
    # waiting frame
    ret, frame = cap.read()
    # mirror the frame vertically
    frame=cv2.flip(frame,1)
    # show text on the frame
    cv2.putText(frame,
                'Press "Q" to Start!!',
                (100, 50), #text pos
                cv2.FONT_HERSHEY_DUPLEX, #font
                1.5, #fontscale
                color, #rgb color
                2, #thickness
                cv2.LINE_AA)
    # display the frame
    frame = cv2.rectangle(frame, (start[0], start[1]), (end[0], end[1]), color, 5)

    # Naming window
    cv2.namedWindow('Window', cv2.WINDOW_NORMAL)

    # Resizing window
    cv2.resizeWindow('Window', 700, 700)

    # Moving window
    cv2.moveWindow('Window', 20, 20)

    cv2.imshow('Window', frame)

    if cv2.waitKey(10) == ord('q'):
        break

```

```
while True:
    # Initializing few auxiliary lists
    data_aux=[]

    ret, frame=cap.read()
    frame=cv2.flip(frame,1)
    cut_frame=frame[start[1]:end[1],start[0]:end[0]]
    frame_rgb=cv2.cvtColor(cut_frame, cv2.COLOR_BGR2RGB)
    results=hands.process(frame_rgb)
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                cut_frame, # Image to draw
                hand_landmarks, # Model output
                mp_hands.HAND_CONNECTIONS, # 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 # X coordinate

                y=hand_landmarks.landmark[i].y # Y coordinate

                data_aux.append(x)
                data_aux.append(y)

        '''
        We have trained the RandomForestClassfier with 42 features
        This means we have used only 1 hand in frame.
        So while predicting if we use more than 1 hand
        then it will throw an error.
        To avoid that, the data_aux list
        that stores the number of features is sliced to 42 elements
        '''
        # Prediction is a list of 1 item
        prediction=model.predict([np.asarray(data_aux[:42])])

        # Predicted Character
        predicted_char=labels_dict[int(prediction[0])]

        # Now lets display this character on the frame
        cv2.putText(frame,
                    predicted_char,# To display
                    (150, 60), # Text pos

                    cv2.FONT_HERSHEY_DUPLEX, # Font
                    1.5, # Fontscale
                    color, # rgb color
                    2, # Thickness
                    cv2.LINE_AA)

    frame = cv2.rectangle(frame, (start[0], start[1]), (end[0], end[1]), color, 5)

    # Naming window
    cv2.namedWindow('Window', cv2.WINDOW_NORMAL)

    # Resizing window
    cv2.resizeWindow('Window', 700, 700)

    # Moving window
    cv2.moveWindow('Window', 20, 20)

    cv2.imshow('Window',frame)

    #cv2.moveWindow('frame', 30, 50)
    if cv2.waitKey(10) == ord('e'):
        break

```

TESTING COMMAND RECOGNITION

```python
import cv2
import mediapipe as mp
import pickle
import numpy as np
import webbrowser
import pywhatkit
import wikipedia
import pyttsx3
from pynput.keyboard import Controller, Key
from youtube_search import YoutubeSearch
from pytube import YouTube
import pygame
import os
from bs4 import BeautifulSoup
import requests

# Different mediapipe objects to show landmarks
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)
keyboard = Controller()
```

```python

# Loading/Unpickling our model
with open('newmodel.p', 'rb') as f:
    model_dict = pickle.load(f)
    f.close()

model = model_dict['model']

# Create a labels dictionary
labels_dict = {0: 'A',1: 'B',2: 'C',3: 'D',4: 'E',5: 'F',6: 'G',7: 'H',8: 'I',
               9: 'J',10: 'K',11: 'L',12: 'M',13: 'N',14: 'O',15: 'P',16: 'Q',
              17: 'R',18: 'S',19: 'T',20: 'U',21: 'V',22: 'W',23: 'X',
              24: 'Y',25: 'Z'}


def wikipedia_info(formed_word):
    try:
        wikipedia.set_lang("en")  # Set Wikipedia language to English
        summary = wikipedia.summary(formed_word, sentences=2)  # Get summary of the formed word
        print("Wikipedia Summary:", summary)

        # Initialize the text-to-speech engine
        engine = pyttsx3.init()
```

```python
        engine.setProperty('rate', 150)  # Speed of speech

        # Read aloud the summary
        engine.say(summary)
        engine.runAndWait()
    except wikipedia.exceptions.PageError:
        print("Wikipedia page not found for the word:", formed_word)
    except wikipedia.exceptions.DisambiguationError:
        print("Ambiguous term. Please provide more specific word.")
    except Exception as e:
        print("Error:", str(e))


def youtube_video(formed_word):
    try:
        results = YoutubeSearch(formed_word, max_results=1).to_dict()
        if results:
            video_url = "https://www.youtube.com" + results[0]['url_suffix']
            webbrowser.open(video_url)
            print("Playing video for:", formed_word)
        else:
            print("No video found for:", formed_word)
    except Exception as e:
```

```
        print("Error playing YouTube video:", str(e))


def read_weather():
    try:
        url = "https://www.accuweather.com/en/in/kolkata/206690/weather-forecast/206690"
        response = requests.get(url)
        soup = BeautifulSoup(response.content, "html.parser")
        weather_info = soup.find("div", class_="CurrentConditions--
phraseValue--2xXSr").text.strip()

        # Initialize the text-to-speech engine
        engine = pyttsx3.init()
        engine.setProperty('rate', 150)  # Speed of speech

        # Read aloud the weather info
        engine.say(f"The current weather in Kolkata is {weather_info}")
        engine.runAndWait()

    except Exception as e:
        print("Error reading weather information:", str(e))


```

```
start=[20,80]
end=[300,380]
color=(255, 0, 127) #bgr format

cap=cv2.VideoCapture(0)

###WAITING WINDOW----------------------
while True:
    # waiting frame
    ret, frame = cap.read()
    # mirror the frame vertically
    frame=cv2.flip(frame,1)
    # show text on the frame
    cv2.putText(frame,
                'Press "Q" to Start!!',
                (100, 50), #text pos
                cv2.FONT_HERSHEY_DUPLEX, #font
                1.5, #fontscale
                color, #rgb color
                2, #thickness
                cv2.LINE_AA)
    # Display rectangle
    frame = cv2.rectangle(frame, (start[0], start[1]), (end[0], end[1]), color, 5)
```

```


    # Naming window
    cv2.namedWindow("Resized_Window", cv2.WINDOW_NORMAL)

    # Resizing window
    cv2.resizeWindow("Resized_Window", 700, 700)

    # Moving window
    cv2.moveWindow("Resized_Window", 20, 20)

    # Displaying the frame
    cv2.imshow("Resized_Window", frame)

    if cv2.waitKey(25) == ord('q'):
        break

# Empty list to store predicted letters
pred_letters = []
command=set()

###COMMAND PREDICTION WINDOW----------------------
while True:
    #command prediction frame
```

```
    data_aux=[]

    ret, frame=cap.read()
    frame=cv2.flip(frame,1)
    cut_frame=frame[start[1]:end[1],start[0]:end[0]]
    frame_rgb=cv2.cvtColor(cut_frame, cv2.COLOR_BGR2RGB)
    results=hands.process(frame_rgb)
    if results.multi_hand_landmarks:

        for hand_landmarks in results.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x=hand_landmarks.landmark[i].x # X coordinate
                y=hand_landmarks.landmark[i].y # Y coordinate

                data_aux.append(x)
                data_aux.append(y)

        # Prediction is a list of 1 item
        prediction=model.predict([np.asarray(data_aux[:42])])

        # Predicted Character
        pred_char=labels_dict[int(prediction[0])]
        pred_letters.append(pred_char)
```

```


        # Now lets display this character on the frame
        cv2.putText(frame,
                    pred_char,# To display
                    (150, 60), # Text pos
                    cv2.FONT_HERSHEY_DUPLEX, # Font
                    1.5, # Fontscale
                    color, # rgb color
                    2, # Thickness
                    cv2.LINE_AA)

    frame = cv2.rectangle(frame, (start[0], start[1]), (end[0], end[1]), color, 5)

    # Naming window
    cv2.namedWindow("Resized_Window", cv2.WINDOW_NORMAL)

    # Resizing window
    cv2.resizeWindow("Resized_Window", 700, 700)

    # Moving window
    cv2.moveWindow("Resized_Window", 20, 20)

```

```
    # Displaying the frame
    cv2.imshow("Resized_Window", frame)

    # Check if specific gesture is performed (e.g., fist close)
    if len(pred_letters) >= 5:  # Minimum 5 letters required
        command = set(pred_letters[-5:]) # Convert last five letters to set
    if (command == {'W'} or command=={'C'} or command=={'B'} or command == {'K'}):
        #command recognized so exit while
        break

    if cv2.waitKey(25) == ord('e'):
        break

print('Command Recognized=',command)

###WEATHER COMMAND-------------------------------

if command == {'B'}:
    # Read weather info aloud
    read_weather()

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

```
207
208 ###WIKIPEDIA OR YOUTUBE COMMAND ---------------------------------
209
210 if command == {'W'} or command == {'C'}:
211     pred_letters = []
212     formed_word=""
213     while True:
214         #frame for forming words to give as input to command
215         data_aux=[]
216
217         ret, frame=cap.read()
218         frame=cv2.flip(frame,1)
219         cut_frame=frame[start[1]:end[1],start[0]:end[0]]
220         frame_rgb=cv2.cvtColor(cut_frame, cv2.COLOR_BGR2RGB)
221         results=hands.process(frame_rgb)
222         if results.multi_hand_landmarks:
223
224             for hand_landmarks in results.multi_hand_landmarks:
225                 for i in range(len(hand_landmarks.landmark)):
226                     x=hand_landmarks.landmark[i].x # X coordinate
227                     y=hand_landmarks.landmark[i].y # Y coordinate
228
229                     data aux.append(x)
230                     data_aux.append(y)
231
232             # Prediction is a list of 1 item
233             prediction=model.predict([np.asarray(data_aux[:42])])
234
235             # Predicted Character
236             pred_char=labels_dict[int(prediction[0])]
237             pred_letters.append(pred_char)
238
239             formed_word = "".join(set(pred_letters))
240
241         # Now lets display this character on the frame
242         cv2.putText(frame,
243                     formed_word,# To display
244                     (20, 60), # Text pos
245                     cv2.FONT_HERSHEY_DUPLEX, # Font
246                     1.5, # Fontscale
247                     color, # rgb color
248                     2, # Thickness
249                     cv2.LINE_AA)
250         frame = cv2.rectangle(frame, (start[0], start[1]), (end[0], end[1]), color, 5)
251
252         # Naming window
253         cv2.namedWindow("Resized_Window", cv2.WINDOW_NORMAL)
254
255         # Resizing window
256         cv2.resizeWindow("Resized_Window", 700, 700)
257
258         # Moving window
259         cv2.moveWindow("Resized_Window", 20, 20)
260
261         # Displaying the frame
262         cv2.imshow("Resized_Window", frame)
263
264         if cv2.waitKey(25) == ord('d'):
265             formed_word=formed_word[:-1]
266             pred_letters=pred_letters[:-1]
267
268         if cv2.waitKey(25) == ord('c'):
269             formed_word=""
270             pred_letters=[]
271
272         if cv2.waitKey(25) == ord('e'):
273             break
274
275     cap.release()
```

```
    cv2.destroyAllWindows()


    print("Formed Text=",formed_word)

    if command=={'W'}:
        wikipedia_info(formed_word)

    if command=={'C'}:
        youtube_video(formed_word)


###VOLUME INCREASE DECREASE COMMAND--------------------

if command == {'K'}:
    while True:
        ret, frame=cap.read()
        frame=cv2.flip(frame,1)
        cut_frame=frame[start[1]:end[1],start[0]:end[0]]
        frame_rgb=cv2.cvtColor(cut_frame, cv2.COLOR_BGR2RGB)
        results=hands.process(frame_rgb)
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
                for i in range(len(hand_landmarks.landmark)):
                    pass
                for id, lm in enumerate(hand_landmarks.landmark):
                    h, w, c = cut_frame.shape
                    cx, cy = int(lm.x * w), int(lm.y * h)

                    # Index finger
                    if id == 8:
                        cv2.circle(cut_frame, (cx, cy), 15, (255, 0, 255), cv2.FILLED)

                        if cx <= 120:
                            keyboard.press(Key.media_volume_down)
                            keyboard.release(Key.media_volume_down)

                        elif cx >= 200:
                            keyboard.press(Key.media_volume_up)
                            keyboard.release(Key.media_volume_up)


        frame = cv2.rectangle(frame, (start[0], start[1]), (end[0], end[1]), color, 5)

        # Naming window
        cv2.namedWindow("Resized_Window", cv2.WINDOW_NORMAL)

        # Resizing window
        cv2.resizeWindow("Resized_Window", 700, 700)

        # Moving window
        cv2.moveWindow("Resized_Window", 20, 20)

        # Displaying the frame
        cv2.imshow("Resized_Window", frame)

        if cv2.waitKey(25) == ord('e'):
            break

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

REFERENCES

1. (n.d.). Retrieved from MediaPipe Hands Documentation: https://mediapipe.readthedocs.io/en/latest/solutions/hands.html

2. Baidaa M Alsafy, Z. M. (2020). *Multiclass Classification Methods A Review.* University of Thi_Qar, Thi_Qar, Iraq.

3. Latha, B. e. (2023). *Hand Gesture and Voice Assistants.* E3S Web of Conferences. Vol. 399. EDP Sciences.

4. Pedregosa, F. a. (2011). *Scikit-learn: Machine Learning in Python.*

5. Pickling: Python Object Serialization. (n.d.). Retrieved from Python Documentation: https://docs.python.org/3/library/pickle.html

6. Qi, J. (2024). *Computer vision-based hand gesture recognition for human-robot interaction: a review.* Complex & Intelligent Systems 10.1.

7. sklearn.model_selection.GridSearchCV. (n.d.). Retrieved from scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

8. Someshwar, D. (2020). *Implementation of virtual assistant with sign language using deep learning and TensorFlow.* Second international conference on inventive research in computing applications (ICIRCA). IEEE.

9. T. J. Swamy, M. N. (2022). *Voice and Gesture based Virtual Desktop Assistant for Physically Challenged People.* Tirunelveli, India: 6th International Conference on Trends in Electronics and Informatics (ICOEI).

10. Tomasz Grzejszczak, M. K. (2016). *Hand landmarks detection and localization in color images.* Silesian University of Technology: Multimedia Tools and Applications.

11. What is Hyperparameter Tuning? (n.d.). Retrieved from Amazon AWS: https://aws.amazon.com/what-is/hyperparameter-tuning/