***A Industrial Oriented Mini Project Report***

*On*

**Sign Language To Text And Speech Conversion Using Hand Landmarks**

*Submitted in partial fulfilment for the request of Degree B. Tech.*

*In*

***Information Technology***

*By*

*Alampally Sravani [22911A1269]*

*Perka Vaishali [22911A12A3]*

*Tirumalasetti Jyothirmai [22911A12C0]*

*Uppuleethi Varshitha [22911A12C1]*

***Under the guidance of***

Mrs. G. Indira Priyadarshini

Associate Professor



**DEPARTMENT OF INFORMATION TECHNOLOGY**

**VIDYA JYOTHI INSTITUTE OF TECHNOLOGY**

(An Autonomous Institution)

**[Accredited by NAAC & NBA, Approved by AICTE New Delhi & Permanently Affiliated to JNTUH]**

# 2024 – 2025



**Department of Information Technology**

**CERTIFICATE**

This is to certify that the project report entitled **“Sign Language To Text And Speech Conversion Using Hand Landmarks”** submitted by ***Alampally Sravani [22911A1269], Perka Vaishali [22911A12A3], Tirumalasetti Jyothirmai [22911A12C0], Uppuleethi Varshitha [22911A12C1]*** to Vidya Jyothi Institute of Technology(An Autonomous Institution), Hyderabad, in partial fulfilment for the award of the degree of **B. Tech. in Information Technology** a *bonafide* record of project work carried out by us under the supervision of **Mrs. G. Indra Priyadarshini**. The contents of this report, in full or in parts, have not been submitted to any other Institution or University for the award of any degree.

**Internal Guide**

**Mrs. G. Indira Priyadarshini**

**Associate Professor**

**Head of the Department**

**Dr. A. Obulesu**

**Associate Professor**

**External Examiner**

# 

# DECLARATION

We declare that this project report titled **Sign Language To Text And Speech Conversion Using Hand Landmarks** submitted in partial fulfilment for the requirement of the degree of B. Tech. in Information Technology is a record of original work carried out by us under the supervision of **Mrs. G. Indira Priyadarshini**, and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice of reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Alampally Sravani [22911A1269]

Perka Vaishali [22911A12A3]

Tirumalasetti Jyothirmai [22911A12C0]

Uppuleethi Varshitha [22911A12C1]

**Date:**

# ACKNOWLEDGEMENTS

We extend our heartfelt gratitude to **Mrs. G. Indira Priyadarshini**, Associate Professor, for her invaluable mentorship and guidance throughout the course of this project. Her expertise, encouragement, and insightful feedback played a crucial role in shaping our approach and refining our methodology.

We are deeply grateful to **Dr. A. Obulesu**, Head of the Department, and all the faculty members of the department for their constant support and guidance throughout our academic journey.

Our sincere thanks to our beloved **Dean of Accreditation and Rankings, Dr. A. Padmaja Madam**, and **Academic Co-ordinator, Smt. G. Srilatha Madam**, whose unwavering support proved indispensable during the course.

We also express our heartfelt thanks to our **Principal, Dr. A. Srujana madam**, for her encouragement and continuous support throughout this endeavour.

We would like to sincerely thank our **College Management** for providing the necessary infrastructure and facilities that enabled us to carry out our work effectively.

Finally, we would like to express our deepest gratitude to our **parents**, whose unwavering love, guidance, and support have been the foundation of our success and aspirations.

*Alampally Sravani [22911A1269]*

*Perka Vaishali [22911A12A3]*

*Tirumalasetti Jyothirmai [22911A12C0]*

*Uppuleethi Varshitha [22911A12C1]*

Table of Contents

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | **Topic** | | **Page No.** | |
|  | | Abstract | | i | |
|  | | List of Figures | | ii | |
|  | | List of Table | | iii | |
| 1.**Introduction** | | | | 1-7 | |
| 1.1 | | Introduction | | 1-2 | |
| 1.2 | | Problem Statement | | 2-3 | |
| 1.3 | | Existing Systems | | 3-5 | |
| 1.4 | | Proposed System | | 5-6 | |
| 1.5 | | Objectives | | 6-7 | |
| 2. **Literature Survey** | | | | 8-18 | |
| 2.1 | | Related Work | | 8-17 | |
| 2.2 | | Research Gap | | 17-18 | |
| 3. **System Analysis** | | | | 19- | |
| 3.1 | | Feasibility Study | | 19-20 | |
| 3.2 | | Software Requirements Specification | | 20-22 | |
|  | | 3.2.1 | Functional Requirements | 21 | |
|  | | 3.2.2 | Non-Functional Requirements | 21-22 | |
| 3.3 | | Software & Hardware Requirements | | 22-24 | |
|  | 3.3.1 | | Software Requirements | 22-23 | |
|  | 3.3.2 | | Hardware Requirements | 24 | |
| 4. **System Design** | | | | 25-29 | |
| 4.1 | | Architecture of the System | | 25 | |
| 4.2 | | UML Diagrams | | 26-28 | |
|  | 4.2.1 | | Use Case Diagram | 26 | |
|  | 4.2.2 | | Class Diagram | 27 | |
|  | 4.2.3 | | Sequence Diagram | 28 | |
| 4.3 | | Module Design | | 29 | |
| 5.**Implementation And Results** | | | | 30-43 |
| 5.1 | | Methodology | | 30-32 | |
| 5.2 | | Sample Code | | 33-41 | |
| 5.3 | | Output Screenshots | | 42-43 | |
| 6. **Testing and Validation** | | | | 44-47 |
| 6.1 | | Introduction | | 44-45 | |
| 6.2 | | Test Case Scenarios | | 46-47 | |
| 7. **Conclusion and Future Enhancement** | | | | 48-51 |
| 7.1 | | Conclusions | | 48-49 | |
| 7.2 | | Future Enhancements | | 49-51 | |
| 8. **References** | | | | 52-53 |

**ABSTRACT**

Communication is a fundamental human need, yet individuals with hearing or speech impairments often face challenges in expressing themselves effectively to those unfamiliar with sign language. This system, *Sign Language to Text and Speech Conversion using Hand Landmarks*, aims to bridge this communication gap by converting American Sign Language (ASL) hand gestures into both text and spoken words in real time. The system leverages computer vision and machine learning techniques to detect and interpret hand gestures captured via a webcam. Using OpenCV for video processing and MediaPipe for real-time hand tracking, the application accurately identifies static ASL gestures corresponding to alphabet letters. Each gesture is processed to determine the position of hand landmarks, which are then analyzed using predefined rules to classify individual letters. These letters are appended to form complete words or sentences, which are displayed on-screen and optionally converted to speech using a text-to-speech engine (pyttsx3). An interactive user interface allows users to control the flow of the application through mouse clicks, including options to predict a gesture, delete a character, clear the sentence, or vocalize the output. The solution runs in real time without the need for external sensors or specialized hardware, making it low-cost, accessible, and user-friendly. This system demonstrates a practical approach to improving accessibility for the deaf and hard-of-hearing community, with future scope for expanding into dynamic gestures, full-sentence recognition, and multilingual speech output.

i

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **FIGURE No.** | **TITLE** | **PAGE No.** |
| 4.1 | Architecture of the System Diagram | 25 |
| 4.2.1 | Class Diagram | 26 |
| 4.2.2 | Use Case Diagram | 27 |
| 4.2.3 | Sequence Diagram | 28 |
| 4.3 | Module Design | 29 |
| 4.3 | Output Screenshots | 38-39 |

ii

**List of Tables**

|  |  |  |
| --- | --- | --- |
| **TABLE No.** | **TITLE** | **PAGE No.** |
| 6.2 | Test Cases | 46-47 |

iii

1. **INTRODUCTION**
   1. **Introduction**

Communication is the cornerstone of human society. It allows people to express their thoughts, feelings, desires, and needs. From verbal dialogue and written language to facial expressions and gestures, the methods of communication are diverse and deeply integrated into daily life. However, for individuals with hearing and speech impairments, traditional methods such as speaking or writing are not always accessible or practical. For these individuals, signlanguage serves as a vital communication tool.

Sign language is a comprehensive, visually-based language composed of hand signs, gestures, facial expressions, and body movements. Among the various types of sign languages, American Sign Language (ASL) is widely recognized and used across North America. ASL provides a complete linguistic system capable of expressing complex thoughts and emotions.

Despite its utility, sign language is not commonly understood by the general population. This creates a communicationbarrier that affects millions of deaf and hard-of-hearing individuals. In everyday situations—whether in a classroom, hospital, workplace, or public space—people who rely on sign language often face difficulties interacting with others who do not understand their language. This communication gap contributes to social isolation, exclusion, and limited access to services.

In response to this issue, researchers and developers have turned to technologicalinnovations. With the rise of artificial intelligence (AI), computer vision, and machine learning, it is now possible to create tools that bridgethe communication gap between sign language users and the hearing population. The focus of this system is to develop a real-time ASL alphabet recognition system that translates hand gestures into readable text and audible speech, thereby making communication smoother and more inclusive.

The system is designed using only a standard webcam and open-source libraries, eliminating the need for expensive hardware. It recognizes static hand gestures corresponding to ASL alphabets and displays the interpreted character on a user interface. The system also converts the recognized text into speech, facilitating verbal communication.

By making sign language recognition more accessible, this system aspires to empower deaf individuals, enhance their independence, and promote inclusivity in everyday interactions.

* 1. **Problem Statement**

Although sign language has been a beacon of communication for individuals with hearing impairments, it remains an unfamiliar territory for much of the hearing population. This disparity results in daily challenges and, at times, serious communication failures for those who rely on ASL. The following points highlight the underlying challenges:

### **Limited Accessibility in Society**

The most significant problem is the lack of widespread knowledge of sign language. In environments such as hospitals, schools, workplaces, and public service centers, communication can become incredibly challenging for individuals using sign language. Not everyone has access to trained sign language interpreters, which forces the deaf community to depend on written communication or the help of others. This can cause delays, misunderstandings, and emotional frustration.

### **Dependence on Human Interpreters**

Hiring interpreters for daily interactions is neither convenient nor cost-effective. For most users, especially in spontaneous situations, interpreter services are unavailable or unaffordable. This dependency creates a level of reliance that restricts their independence in various aspects of life.

### **High Cost and Complexity of Existing Solutions**

Many existing technologies attempt to solve the sign language recognition problem using expensive or specialized hardware. These solutions include wearable gloves with sensors or 3D cameras, which may provide accurate readings but are out of reach for most users due to their high costs and setup complexity. These devices are also not suitable for mobile use or on-the-go communication.

### **Lack of Real-Time Performance**

Even with powerful tools available, most systems do not operate in real-time. Delays in gesture recognition and translation disrupt the natural flow of communication, making it difficult to maintain interactive conversations. For such systems to be truly useful, they must process and interpret gestures instantly with minimal latency.

### **Lack of Natural User Interfaces**

### Systems that require physical equipment, such as gloves or additional sensors, make the experience less intuitive. Users are less likely to adopt tools that demand physical attachments or pose learning curves. A natural interface using a regular webcam allows for seamless interaction that mimics how gestures are normally made in daily life.

**1.3 Existing Systems**

Several existing systems have been developed to tackle the problem of sign language recognition, each employing distinct technological approaches. However, while these systems contribute to advancements in the field, they also come with practical limitations that restrict their widespread usability—particularly in resource-constrained environments.

**Glove-based systems** utilize gloves embedded with motion or flex sensors to detect hand and finger movements with high accuracy. Despite their precision, these systems are often expensive, bulky, and uncomfortable for long-term use. They also require users to wear hardware, which may not be feasible or desirable in daily communication scenarios. These factors make glove-based solutions impractical for casual or large-scale deployment.

**Image processing systems**, which use standard webcams along with techniques such as contour detection and skin segmentation, offer a more affordable and less intrusive solution. However, these systems tend to be highly sensitive to environmental conditions. Variations in lighting, background clutter, and hand orientation can significantly degrade their performance. This dependency on ideal external conditions makes them less robust for real-world usage.

**Depth camera-based systems**, such as those utilizing Microsoft Kinect or Intel RealSense, incorporate depth sensing to enhance the segmentation and recognition of hand gestures. While these systems improve accuracy and can recognize gestures in 3D space, their dependency on expensive and specialized hardware makes them unsuitable for everyday use by the average user or in low-budget institutions.

**Neural network-based systems** leverage advanced machine learning models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to classify both static and dynamic gestures. These systems demonstrate high accuracy, especially when trained on large datasets. However, they require extensive labeled data, significant training time, and high computational power—often involving GPUs or cloud-based infrastructure. This makes real-time, offline deployment difficult, especially on devices with limited processing capabilities.

**Mobile application-based systems** attempt to offer accessibility by using built-in smartphone cameras and lightweight AI models to detect gestures. While they provide convenience and portability, their gesture recognition capabilities are often limited due to constraints in hardware, battery life, and processing power. Moreover, small screen sizes and inconsistent camera angles can hinder usability and accuracy.

**Limitations of Existing Systems**

* **Glove-based Systems:**
  + Require users to wear external hardware
  + High cost and low comfort for everyday use
  + Not feasible for casual or large-scale adoption
* **Image Processing Systems:**
  + Highly sensitive to lighting and background conditions
  + Reduced accuracy with hand occlusions or orientation changes
  + Poor robustness in dynamic or uncontrolled environments
* **Depth Camera-based Systems:**
  + Depend on costly and specialized hardware
  + Limited portability
  + Unfit for mobile or low-resource applications
* **Neural Network-based Systems:**
  + Require massive computational resources and training data
  + Not suited for real-time processing on consumer devices
  + Offline deployment is difficult without powerful hardware
* **Mobile Application-based Systems:**
  + Limited by smartphone hardware and battery life
  + Reduced recognition accuracy due to small screen and camera angles
  + Can become sluggish under sustained usage or poor lighting

**1.4 Proposed System**

The proposed system is designed to recognize static hand gestures corresponding to the American Sign Language (ASL) alphabet and convert them into text and speech in real time. It provides an accessible, low-cost, and efficient solution for assisting communication between hearing-impaired individuals and the general public.

Unlike many traditional approaches that rely on expensive hardware such as sensor gloves, depth cameras, or advanced machine learning models, this system adopts a rule-based recognition technique. It uses a standard webcam and MediaPipe’s real-time hand landmark detection to identify gestures based on the geometric relationships of 21 hand points. These landmarks are analyzed using conditional rules to recognize specific alphabet gestures.

Once a gesture is recognized, the system displays the corresponding letter, constructs full sentences by appending each recognized character, and optionally converts the text into speech using a built-in text-to-speech (TTS) engine (pyttsx3). The graphical user interface is built using OpenCV and allows user-friendly interaction through buttons like Predict, Clear, Delete, Space, and Speak.

This system operates entirely offline and is suitable for low-resource environments such as schools, rural centers, or individuals’ homes, making it ideal for broad and inclusive deployment.

**Advantages of the Proposed System**

* **Cost-Effective**  
  Utilizes only a webcam and open-source software libraries. No additional sensors, gloves, or GPUs are required.
* **Real-Time Recognition**  
  Processes gestures instantly and updates the text field in real time, providing immediate feedback to the user.
* **Offline Functionality**  
  The system works without an internet connection, ensuring usability in areas with limited or no connectivity.
* **User-Friendly Interface**  
  Offers a simple and intuitive graphical interface with easily accessible controls for all user types, including beginners.
* **No Training Data Required**  
  Unlike machine learning models, this rule-based approach eliminates the need for datasets, model training, or labeling.
* **Easy to Extend**  
  New gestures or additional rules can be incorporated into the system logic without retraining, making it flexible for updates.
* **Inclusive Communication**  
  Bridges the gap between sign language users and non-signers by offering both visual (text) and auditory (speech) outputs.
* **Platform Independence**  
  Developed in Python using portable libraries like OpenCV and MediaPipe, the system can run on multiple platforms with minimal configuration.

**1.5 Objectives**

The primary objective of this system is to design and implement a real-time, webcam-based system that recognizes American Sign Language (ASL) static alphabet gestures and converts them into both textual and spoken output. This system aims to bridge the communication gap between sign language users and non-signers through a simple and cost-effective interface.

**Specific Objectives:**

* **To develop a vision-based system** capable of capturing hand gestures using a standard webcam without requiring any specialized hardware.
* **To recognize static ASL alphabet gestures** by analyzing hand landmarks using geometric rules rather than relying on machine learning models or training datasets.
* **To construct full words and sentences** from sequentially recognized gestures and allow users to edit or clear text in real time.
* **To integrate a text-to-speech (TTS) engine** that audibly conveys the constructed sentence to facilitate two-way communication.
* **To ensure offline operability** so that the system can function without an internet connection, making it suitable for rural or low-resource environments.
* **To design an intuitive and accessible user interface** using OpenCV that provides controls like Predict, Clear, Delete, Space, and Speak.
* **To build a modular and scalable system architecture** that can be easily extended to include new gestures, support dynamic signs, or adapt to other sign languages in future versions.

**2.Literature Survey**

Choosing the right approach for sign language recognition has become more complex due to the rapid development of technologies in computer vision and assistive tools. Traditional systems often relied on costly hardware or lacked flexibility and real-time performance, limiting their practical use.

This literature survey reviews previous methods used for sign language recognition, including sensor-based devices, image processing, and neural networks. It focuses on the technologies adopted, their applications, and the common challenges they faced, helping to understand the progress and gaps in this field.

## Related Work

## a) Translation of Sign Language Finger-Spelling to Text using Image Processing

**Authors:** Krishna Modi and Amrita More B.Tech. (Computer), Mukesh Patel School of Technology and Management Engineering, Mumbai

**Summary:**

The paper presents a practical and efficient system that translates American Sign Language (ASL) finger-spellings into English text using only a standard webcam and image processing techniques. Unlike more complex systems that require gloves or depth cameras, this method relies on basic computer vision techniques like grayscale conversion, binary thresholding, and BLOB (Binary Large Object) analysis. The system operates in real-time, capturing hand gestures and comparing them to a pre-built database of ASL alphabet gestures. The outcome is displayed as corresponding English alphabets, which can then be used to construct full words and sentences. With an average recognition accuracy of 96%, the system proves to be a valuable tool for deaf-mute communication.

**Objective:**

The main objective of this research is to design a simple, affordable, and effective method for translating sign language finger-spellings (specifically ASL) into text. The authors aim to eliminate the need for expensive hardware or complex algorithms and instead propose a webcam-based system that uses standard image processing techniques to facilitate communication for the deaf-mute community.

**Methodology:**

The methodology is divided into several clearly defined stages:

* **Video Capture:**
  + - The system captures video using an internal or external webcam.
    - A single frame is extracted every 4 seconds to give users time to form a gesture.
* **Smoothing:** The input image is smoothened using a mean filter to reduce noise.
* **Grayscale Conversion:** RGB image is converted using weighted sums (0.2125R + 0.7154G + 0.0721\*B).
* **Binary Thresholding:** The image is binarized using a threshold value (set to 40) to isolate the hand from a black background.
* **Mirroring (if needed):** Left-hand gestures are mirrored to match the right-hand-based database.
* **Image Comparison:** Each processed image is compared pixel-by-pixel with three stored samples for each ASL alphabet in the database. The percentage of matching pixels is computed.
* **Gesture Recognition:** The gesture is recognized if the match exceeds 75%. The corresponding alphabet is displayed.
* **Word Formation:** Recognized alphabets are concatenated to form words like “HELLO,” “WORLD,” and “YELLOW.”
* **Tools Used:** AForge.NET library in C# (Visual Studio 2010) for image processing tasks.

**Key Findings:**

* Achieved 96% recognition accuracy for ASL finger-spellings.
* Recognition time was extremely low — average of 50 milliseconds per frame.
* The system is fully functional with a standard webcam; no gloves or depth cameras required.
* Supports both right-handed and left-handed users through mirroring.
* Letters J and Z were excluded due to their dynamic nature.
* Accurate word formation demonstrated for multiple example words.
* Simple pixel-by-pixel comparison yielded robust results with minimal processing.

**Implications:**

This system has strong implications for real-world communication enhancement for the deaf and mute. Its simplicity makes it accessible for educational use, personal communication, and integration into low-cost assistive devices. Moreover, its independence from specialized hardware makes it suitable for widespread deployment, especially in resource-constrained environments like rural schools or public service offices. The system can serve as a foundational model for future expansions, such as adding support for dynamic gestures, incorporating audio output, or integrating with mobile platforms.

**Conclusion:**

The research concludes that the proposed system is a promising and effective method for recognizing and translating ASL finger-spellings into English text. The high accuracy rate (96%) and fast response time (50 milliseconds) make it suitable for real-time use. The authors emphasize the simplicity and accessibility of their approach, which eliminates the need for expensive equipment or complex machine learning models. The system’s design allows for potential expansion, such as support for motion gestures and full sentence translation. Overall, this system contributes significantly to assistive technology for the deaf-mute community and demonstrates the power of classical image processing techniques in real-world applications.

**b) Hand Gesture Recognition for Human-Machine Interaction**

**Authors:** Elena Sánchez-Nielsen, Luis Antón-Canalís, and Mario Hernández-Tejera

**Summary:**

This paper presents a system for recognizing static hand gestures for human-machine interaction, with the goal of enabling intuitive communication between users and computers. The authors developed a technique that processes color images to detect skin-toned regions, extract hand BLOBs (Binary Large Objects), and compare them with a pre-defined database using the Hausdorff distance algorithm. The system achieved 90% recognition accuracy across 26 alphabet gestures, demonstrating its effectiveness for controlled environments. The paper highlights the strength of combining geometric matching with color-based segmentation but acknowledges the computational complexity involved.

**Objective**

The main objective of this paper was to design a system capable of recognizing static hand gestures that correspond to 26 alphabet letters used in sign language. The purpose was to create a method of human-machine interaction where users can communicate with computers or other devices using hand gestures, enabling more natural, intuitive interaction for both general and impaired users.

**Methodology**

The method implemented in this work involves capturing color images and applying color space transformations to accurately detect skin-colored regions in the input. Once detected, these skin-like regions are processed to extract the most relevant Binary Large Object (BLOB), assumed to be the hand. After isolating the hand gesture from the image, the system applies the Hausdorff distance algorithm to compare the captured gesture shape with those stored in a database. This comparison helps in identifying which letter of the alphabet the gesture corresponds to. The process includes multiple image processing steps such as skin detection, noise filtering, and geometric analysis.

**Key Findings**

* **High Recognition Accuracy:** The system achieved a 90% recognition accuracy in identifying 26 static hand postures of the alphabet, indicating strong reliability in controlled environments.
* **Effective Use of Color Space Transformation:** By converting color images to a specific color space for skin detection, the system could effectively isolate hand regions for further analysis.
* **BLOB Extraction for Hand Isolation:** The use of Binary Large Object (BLOB) detection helped accurately identify the hand in an image, reducing background noise and improving the precision of recognition.
* **Hausdorff Distance for Shape Matching:** The application of the Hausdorff distance algorithm enabled precise comparison between the input gesture and stored templates, contributing significantly to the classification accuracy.
* **Suitable for Static Gestures:** The method was well-suited for static gestures, though it lacked the ability to handle dynamic or continuous gesture inputs efficiently.

**Implications**

This research demonstrated that computer vision techniques, combined with shape-based comparison methods like Hausdorff distance, can be effectively used in static gesture recognition systems. It emphasized the viability of developing HCI (Human-Computer Interaction) applications for sign language recognition without the need for specialized equipment. However, the relatively complex mathematical computation required by the Hausdorff distance algorithm could pose a limitation for real-time applications on devices with low computational resources.

**Conclusion**

While effective in terms of accuracy, the approach used in this paper involves a relatively complex and computationally intensive algorithm. This limits its scalability and real-time performance, especially in low-power environments. Nonetheless, it set a strong foundation for future gesture recognition systems by combining reliable image processing with precise geometric shape comparison.

**c) An Image Processing Technique for the Translation of ASL Finger-Spelling to Digital Audio or Text**

**Authors:** Chance M. Glenn, Divya Mandloi, Kanthi Sarella, Muhammed Lonon

(The Laboratory for Advanced Communications Technology, Rochester Institute of Technology)

**Summary:**

This paper introduces a robust image-processing-based system to translate ASL finger-spellings into either text or digital audio. The method uses adaptive databases that evolve as more gestures are recorded, making the system more reliable and personalized over time. It avoids the use of external sensors by relying entirely on webcam-captured images.

**Objective:**

The primary aim of this work was to develop a system that can translate American Sign Language (ASL) finger-spellings into text or digital audio, with the goal of enhancing accessibility for deaf-mute individuals. The focus was on creating a more efficient communication method by eliminating the need for interpreters, enabling direct interaction through visual gestures.

**Methodology:**

The system captures real-time video input and processes individual frames by isolating the hand region using RGB color filtering and thresholding to remove the background. The technique incorporates edge detection to highlight hand contours, and the gesture is cropped and resized to a standardized size. One of the key features of the system is its use of an adaptive statistical database that learns from the data it receives, which continuously adjusts as more gestures are processed. This allows the system to adapt to new input over time, improving its accuracy and functionality. The recognition process uses an error matrix to compare pixel-level differences between the input image and the images stored in the database, providing an effective recognition mechanism.

**Key Findings:**

* The system was successful in translating ASL finger-spellings into text or audio with real-time efficiency.
* It showed flexibility by updating its statistical database, adapting to new gestures and variations over time.
* While a specific accuracy figure was not provided, the approach was noted for its ability to perform dynamically, offering practical application potential in real-world environments where gestures may vary.

**Implications:**

The paper emphasizes the importance of adaptive learning in gesture recognition systems. The ability to continuously update the system’s database ensures that it can handle slight variations in gestures, which is essential for real-world usage. Additionally, the system's dynamic learning capability reduces the need for frequent manual updates, making it more scalable and self-improving. This innovation points towards the future potential for creating systems that require minimal intervention while improving over time.

**Conclusion:**

The work presented a functional, real-time solution for ASL finger-spelling translation, with a focus on adaptive learning. This approach was influential for the authors of the current system, especially the integration of an evolving database. However, the goal for the present system is to streamline the process, making it more implementable and stepwise in nature. This paper serves as both a benchmark and partial blueprint for their ongoing work, laying a solid foundation for future advancements in gesture recognition.

**d) Real-Time Hand Gesture Recognition Using a Range Camera**

**Authors:** Zhi Li and Ray Jarvis (Monash University, Australia)

**Summary:**

This paper explores the use of range cameras (depth sensors) for real-time hand gesture recognition. Unlike RGB-only methods, it leverages depth data to more accurately isolate the hand from complex backgrounds. The system is designed to detect both static and dynamic gestures, enabling its use in real-time applications such as virtual interfaces and robotics.

**Objective**

The objective of this study was to utilize range (depth) camera technology to enhance the accuracy of real-time hand gesture recognition systems. The researchers aimed to improve the detection of hand gestures by leveraging depth information in addition to standard 2D images, thereby enabling a more detailed and robust representation of hand shapes and positions.

**Methodology**

This system employed a range (depth) camera, capable of capturing the 3D coordinates of hand movements. The recognition process involved extracting 3D hand models and analyzing them using spatial tracking techniques that measure depth, orientation, and finger articulation. The depth data allows for a clearer distinction between overlapping parts of the hand, which is often a challenge in traditional 2D image processing systems. The system imposed specific spatial constraints and relied on complex real-time processing algorithms to accurately identify gestures.

**Key Findings**

The system demonstrated high accuracy in detecting hand gestures due to the additional dimension of depth, which enhanced its ability to separate hand gestures from the background and resolve occlusions. The use of a range camera significantly improved the robustness of gesture recognition under various lighting and background conditions. However, the system's complexity and reliance on expensive, specialized hardware were notable drawbacks.

**Implications**

While the use of depth sensors can greatly enhance recognition accuracy, it introduces significant cost and complexity. The requirement for specific hardware like a depth camera limits the system's accessibility and scalability, particularly in environments where such resources are not readily available. This makes it more suitable for high-end applications or research settings than for everyday use.

**Conclusion**

Although the system offers advanced recognition capabilities, the authors of the current system did not adopt this approach due to its hardware dependencies and complex processing requirements. It was acknowledged as a powerful technique but considered impractical for their goal of building a low-cost, easily implementable solution that can work with a standard webcam.

**e) Sign Language to Text and Speech Translation in Real Time Using Convolutional Neural Network**

**Authors:** Ankit Ojha, Ayush Pandey, Shubham Maurya, Abhishek Thakur, Dr. Dayananda, Department of Information Science and Engineering, JSS Academy of Technical Education (JSSATE), Bangalore, India

**Summary:**

This paper introduces a real-time desktop application that uses computer vision and deep learning to recognize American Sign Language (ASL) finger-spelling gestures. The system uses a standard webcam to capture hand gestures, processes the image frames using Convolutional Neural Networks (CNN), and translates them into corresponding text and audible speech. By converting the gestures to text and then speech through the pyttsx3 library, the authors aim to bridge the communication gap between hearing-impaired individuals and others. The model achieves a gesture recognition accuracy of approximately 95%, demonstrating the potential of CNNs in real-time gesture recognition applications.

**Objective:**

The primary objective is to develop a real-time system capable of translating ASL finger-spelling into both textual and speech outputs using a CNN-based model. The goal is to create a communication aid for people with hearing impairments that does not require external hardware like gloves or special sensors, but functions effectively using a webcam and image processing.

**Methodology:**

* **Data Collection:** A dataset of 1,000 ASL gesture images was collected, covering alphabets and digits. Each image was processed to a standard dimension of 50×50 pixels in grayscale.
* **Image Acquisition:** Webcam streams are captured using OpenCV. The video feed is divided into frames and each frame is preprocessed for gesture recognition.
* **Preprocessing:** Images undergo grayscale conversion and segmentation to isolate the hand gesture. Gaussian background subtraction is used to enhance the gesture area. Bounding boxes are created to localize hand gestures using Gaussian Mixture Models.
* **CNN Architecture:** Three convolutional layers extract features at increasing complexity levels (lines, angles, shapes). Max pooling layers reduce spatial dimensions. The dense layers map features to gesture classes. Dropout layers prevent overfitting. Final layer outputs prediction probabilities for each of the 44 classes.
* **Gesture Classification:** The label with the highest probability is selected as the predicted gesture. Individual gestures are accumulated to form words and sentences.
* **Text and Speech Output:** Text output is displayed on screen. The text is converted to speech using the pyttsx3 Python library, enabling audible output.

**Key Findings:**

* Achieved an overall gesture recognition accuracy of 95%.
* Successfully translated ASL static finger-spellings into meaningful text and speech.
* The CNN was effective at distinguishing complex hand gestures using low-resolution input.
* The application runs in real-time with minimal delay from webcam capture to speech output.
* The use of OpenCV, Keras, and pyttsx3 ensures cross-platform support and lightweight execution.
* Preprocessing and CNN model are robust enough to handle varied lighting conditions with some adjustments.

**Implications:**

This system demonstrates a practical use of CNNs in creating assistive technologies for the hearing-impaired. Its accessibility—requiring only a webcam and no specialized hardware—makes it scalable and suitable for integration into schools, public services, and home environments. It also opens doors for multilingual sign language translation systems and deeper integrations with mobile and web platforms. Moreover, the implementation highlights how real-time gesture recognition can enhance communication systems in a world moving toward inclusivity.

**Conclusion:**

The study concludes that Convolutional Neural Networks are a powerful tool for solving real-time gesture recognition problems. The proposed application effectively translates ASL finger-spelling into both text and speech with a high degree of accuracy (95%), eliminating the need for human interpreters. While the current version supports only static gestures, the authors acknowledge the importance of incorporating context-based and dynamic gestures in future versions. Potential enhancements include mobile or web-based deployment, multilingual support, and integration with Natural Language Processing (NLP) for full sentence comprehension. Overall, the system marks a significant contribution toward inclusive, real-time communication technology for the deaf and hard-of-hearing community.

**2.2 Research Gap**

**1. Hardware Dependency**

Many existing systems, such as glove-based and depth-camera-based solutions, rely on specialized hardware to detect and interpret hand gestures accurately. Although effective in controlled environments, their high cost and need for additional setup make them unsuitable for widespread or daily use, especially in schools, public sectors, or low-income regions. There is a need for a solution that performs reliably with minimal, accessible hardware—such as a built-in laptop or webcam.

**2. Environmental Sensitivity**

Webcam-based systems that use traditional image processing techniques like color segmentation or thresholding often perform poorly under variable lighting or cluttered backgrounds. This environmental dependency significantly impacts their robustness and usability in real-world, dynamic settings. A system that remains stable and accurate despite lighting changes, skin tone variations, or background interference remains an unmet need.

**3. Computational Complexity**

Several approaches involve shape-matching algorithms (e.g., Hausdorff distance) or adaptive databases that continuously update. While innovative, these methods require complex mathematical operations and higher computational capacity. Their resource-heavy nature makes them inefficient for deployment on low-power devices, posing a barrier to portability and scalability.

**4. Limitations of Deep Learning-Based Approaches**

Although CNN and RNN models offer strong classification capabilities, they come with significant drawbacks. They depend on large, curated datasets for training, require extensive training time, and are prone to overfitting. These models also lack transparency in decision-making and are not easily modifiable for users who want to adapt or extend gesture sets. Retraining such models for customization is time-consuming and not user-friendly for non-technical audiences.

**5. Incomplete Communication Pipeline**

Many systems stop at gesture recognition and fail to provide end-to-end support—such as combining recognition with real-time text display, sentence construction, or speech synthesis. Users are left with incomplete interaction, requiring additional tools or manual steps to convert signs into spoken language. This disrupts natural communication flow and limits usefulness in practical scenarios.

**6. Dependency on Internet or Cloud Services**

Some modern implementations require cloud processing or internet access to handle computation or storage needs, particularly in mobile app-based systems. This makes them unsuitable for areas with poor network coverage and contradicts the goal of making sign recognition universally accessible and usable anytime, anywhere.

**7. Lack of Personalization and Flexibility**

Most existing systems follow rigid, predefined gesture templates or trained models. They do not support user-level customization or learning of unique gesture patterns. A flexible system that can accommodate individual variations, support learning new gestures, or adapt based on usage patterns is still missing in the current literature.

**8. Inability to Handle Multilingual or Regional Sign Languages**

The majority of research focuses on American Sign Language (ASL), with limited consideration for other sign languages like Indian Sign Language (ISL), British Sign Language (BSL), or region-specific adaptations. This limits the global applicability of such systems. There is a noticeable gap in developing multilingual sign language recognition frameworks that can support a broader, diverse user base.

1. **SYSTEM ANALYSIS**

**3.1 Feasibility Study**

A feasibility study serves as a critical preliminary assessment to determine whether a proposed system is viable and worth pursuing. It encompasses various dimensions, including technical, economic, operational, and schedule feasibility. By identifying potential challenges early and validating the core concept, the feasibility study ensures that the development team can plan and implement the system with a higher probability of success.

* **Technical Feasibility**

The technical feasibility of the sign language recognition system is highly favorable. The system is based on Python, one of the most versatile and beginner-friendly programming languages. Python’s extensive ecosystem includes robust libraries such as MediaPipe for hand landmark detection, OpenCV for real-time video stream processing, and pyttsx3 for offline text-to-speech functionality. These libraries are well-documented and widely adopted, reducing the technical risk associated with development.

Moreover, the proposed system does not depend on specialized hardware such as sensor gloves, infrared cameras, or depth sensors. A basic webcam is sufficient for accurate gesture detection. This greatly enhances the system’s deployability and reduces technical complexity. The reliance on rule-based recognition over machine learning models also eliminates the need for GPU acceleration or heavy computational power.

* **Economic Feasibility**

From a financial perspective, the proposed system is cost-effective. Since it is built using open-source libraries and tools, there are no licensing fees or proprietary constraints. Development and deployment require only basic computing infrastructure—most modern laptops meet the system requirements. This reduces the total cost of ownership and makes the system particularly suitable for deployment in educational institutions, government-run accessibility programs, and low-income communities.

Compared to alternative systems that require extensive data collection, model training, or hardware sensors, the economic investment in this system is minimal. Maintenance costs are also low due to the system’s modular and rule-based design, which simplifies debugging and upgrades.

* **Operational Feasibility**

The operational feasibility is determined by how well the system integrates into the user’s environment and supports real-world use cases. This system is designed to function offline, which is critical for users in rural or under-connected areas. The system runs on low-end hardware and offers a simple graphical user interface (GUI) built using OpenCV. Users interact with buttons like Predict, Clear, Delete, Speak, and Space, allowing for seamless control without the need for technical knowledge.

The system's design focuses on inclusivity and accessibility. It supports both left- and right-handed users and recognizes static ASL gestures, making it suitable for beginners or students learning sign language. By offering real-time feedback and audible output, it facilitates smooth interaction between sign language users and non-signers, improving social integration and reducing communication barriers.

* **Schedule Feasibility**

The development timeline for the system is achievable within a standard academic or professional system cycle. Since the architecture is modular and relies heavily on reusable components, development tasks can be parallelized or completed incrementally. Each module—gesture capture, recognition, text generation, and speech output—can be developed and tested independently.

The absence of training datasets and model tuning greatly reduces the time required for implementation. Pre-existing libraries like MediaPipe and OpenCV handle complex operations like landmark detection and GUI rendering, further accelerating development. The system can be prototyped, tested, and iterated within a few weeks, making it suitable for short-term research, capstone systems, or pilot deployments.

**3.2 Software Requirements Specification (SRS)**

The Software Requirements Specification (SRS) provides a comprehensive and structured explanation of what the system is expected to do. It serves as the foundation for all future design, development, and testing activities. The SRS is divided into functional and non-functional requirements, which together define the full scope and constraints of the system.

**3.2.1 Functional Requirements**

Functional requirements describe the core operations and behaviors that the system must support. These include:

* **Webcam Input**: The system shall continuously capture video frames using the built-in or external webcam.
* **Hand Landmark Detection**: The system shall process each frame to identify and track 21 key hand landmarks using MediaPipe.
* **Gesture Recognition**: Based on the spatial configuration of the detected landmarks, the system shall apply rule-based logic to recognize ASL alphabet gestures.
* **Text Generation**: The system shall display the recognized character on the screen and append it to an active sentence.
* **Editing Functions**:
  + **Clear**: Clears the entire constructed sentence.
  + **Delete**: Removes the last character in the sentence.
  + **Space**: Inserts a space between words.
* **Speech Output**: The system shall convert the final sentence to audio using a TTS engine (pyttsx3) when the **Speak** button is pressed.
* **Offline Operation**: The system shall run entirely offline, without requiring any cloud services or internet access.

**3.2.2 Non-Functional Requirements**

Non-functional requirements define system qualities such as performance, reliability, and usability.

* **Performance**: Gesture recognition and character output should occur within 500 milliseconds to ensure real-time feedback.
* **Usability**: The GUI should be intuitive, with large, clearly labeled buttons for easy navigation.
* **Reliability**: The system should accurately detect hand gestures across a range of lighting conditions and hand orientations.
* **Portability**: The application should function on both Windows and Linux platforms.
* **Maintainability**: Codebase should follow modular design principles to facilitate updates and enhancements.
* **Extensibility**: The system should support the addition of new gestures or integration with advanced models in future versions.
* **Security**: Since the system operates locally and does not involve user data or cloud storage, security risks are minimal. Still, basic file and input handling should be robust.

**3.3 Software & Hardware Requirements**

A comprehensive understanding of the software and hardware requirements is critical for ensuring the successful development, deployment, and testing of any intelligent system—especially one focused on real-time sign language recognition and speech conversion. This system is intended to be accessible and functional across a variety of platforms, emphasizing compatibility with commonly available hardware and open-source software tools. Below is a detailed description of the software and hardware components necessary for the efficient operation of the proposed system.

**3.3.1 Software Requirements**

The software environment defines the foundation upon which the system operates. Choosing the right tools ensures optimal performance, easy development, and long-term maintainability. The software stack chosen for this system includes both general-purpose development tools and specialized libraries tailored for real-time gesture recognition and speech processing.

**• Operating System**  
The system is compatible with widely used desktop operating systems, allowing broad accessibility and deployment.

* **Windows 10/11**: These operating systems offer a user-friendly interface, stable performance, and wide support for peripheral devices and drivers.
* **Ubuntu 18.04 or later**: Ubuntu provides a robust open-source alternative, ideal for developers who prefer a Linux-based environment with superior package management and scripting capabilities.

**• Programming Language**

* **Python 3.7 or higher**: Python was chosen due to its readability, extensive support for machine learning and computer vision libraries, and large developer community. It allows quick development and testing of prototypes, as well as smooth integration with AI tools.

**• Python Libraries**

These libraries are crucial for specific system functionalities:

* **MediaPipe**: Developed by Google, MediaPipe provides efficient, lightweight, and real-time hand tracking and landmark detection. It supports cross-platform functionality and integrates seamlessly with OpenCV.
* **OpenCV (Open Source Computer Vision Library)**: OpenCV is essential for capturing real-time video from a webcam, preprocessing image data, and overlaying graphical user interface components such as bounding boxes or text.
* **pyttsx3**: This is a cross-platform text-to-speech conversion library in Python, supporting offline operation. It ensures that the speech output feature remains functional even without an active internet connection.
* **NumPy**: A fundamental package for numerical computations in Python, NumPy supports matrix operations, which are essential for landmark normalization, gesture prediction, and data preprocessing.

**• IDE/Development Tools**  
For efficient coding, debugging, and testing, multiple integrated development environments and tools can be used:

* **Visual Studio Code**: A lightweight yet powerful code editor with extensive Python support and integration with version control tools.
* **PyCharm**: A feature-rich Python IDE designed specifically for Python development. It provides advanced debugging and package management features.
* **Jupyter Notebook**: Especially useful for prototyping and testing code snippets. It supports interactive code execution, making it easier to visualize results and outputs.
* **pip or conda**: These are package managers used to install and manage software dependencies. pip is Python’s built-in package installer, while conda is preferred for managing Python environments in data science workflows.
  + 1. **Hardware Requirements**

In addition to the software stack, the system requires hardware that is capable of real-time video capture, image processing, and speech synthesis. The system is designed to run on affordable and widely available hardware to ensure inclusivity and ease of deployment.

**• Processor**

* **Intel Core i3 (2.0 GHz or above)** or equivalent AMD processors are the minimum requirement to ensure that hand detection and gesture recognition modules run smoothly in real time.
* For optimal performance—especially when running multiple applications simultaneously or processing high-resolution video streams—a faster processor such as Intel i5 or i7 is recommended.

**• RAM**

* **Minimum: 4 GB**: Sufficient for basic development and testing of the system, especially if no other memory-intensive applications are running concurrently.
* **Recommended: 8 GB or higher**: Ensures smoother multitasking, faster processing, and reduced latency in gesture recognition and text-to-speech conversion.

**• Camera**

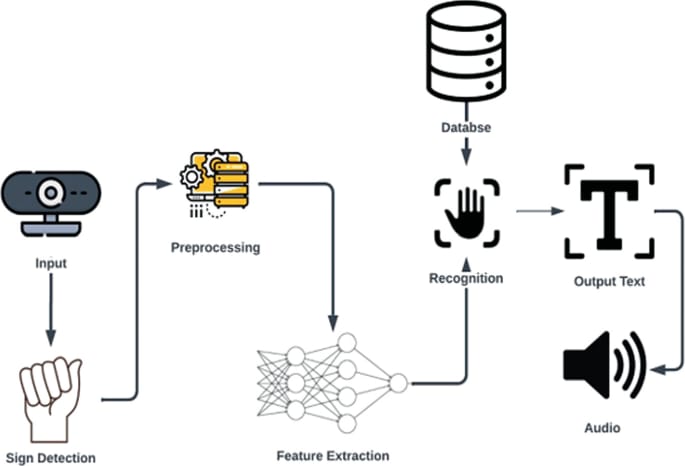
* **Built-in or External Webcam**: A standard 720p webcam is sufficient for capturing hand gestures. Higher resolutions (such as 1080p) improve the accuracy of hand tracking, especially in low-light conditions.
* The camera should support a minimum frame rate of 15–30 frames per second (fps) to provide fluid hand motion capture, which is essential for accurate dynamic gesture recognition.

**• Disk Space**

* **Minimum 100 MB free space**: Required for installing Python libraries, storing intermediate data files, and saving the trained models.
* Additional space may be needed if the application is extended to include data logging or offline training capabilities.

**4.SYSTEM DESGIN**

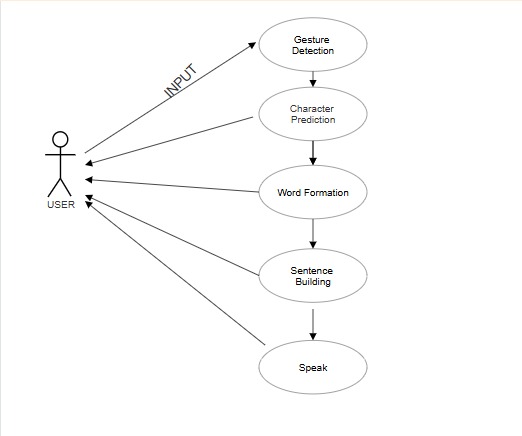
**4.1 Architecture of the System**

****

**Fig(4.1)**

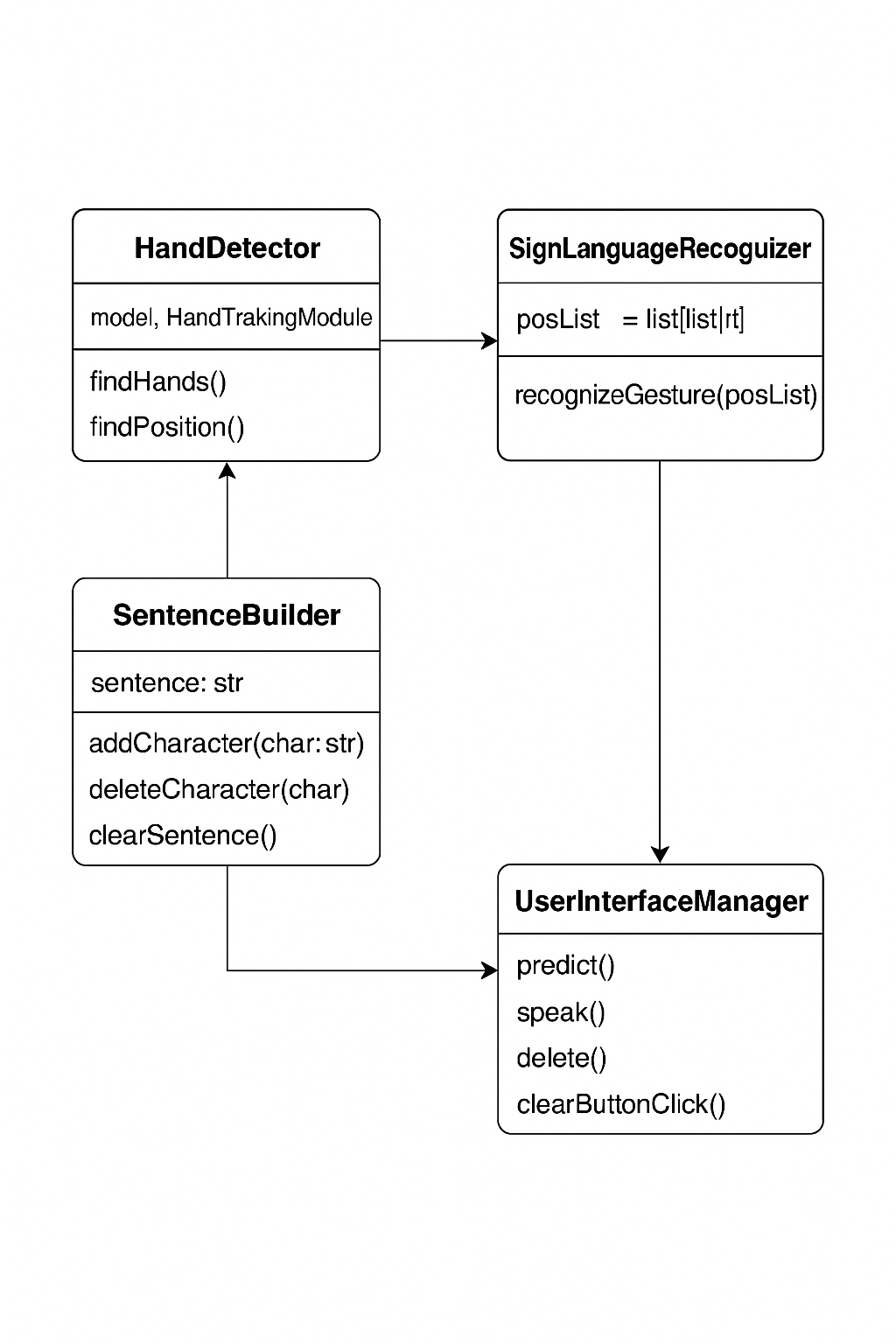
**4.2 UML Diagrams**

**4.2.1 Use Case Diagram**

****

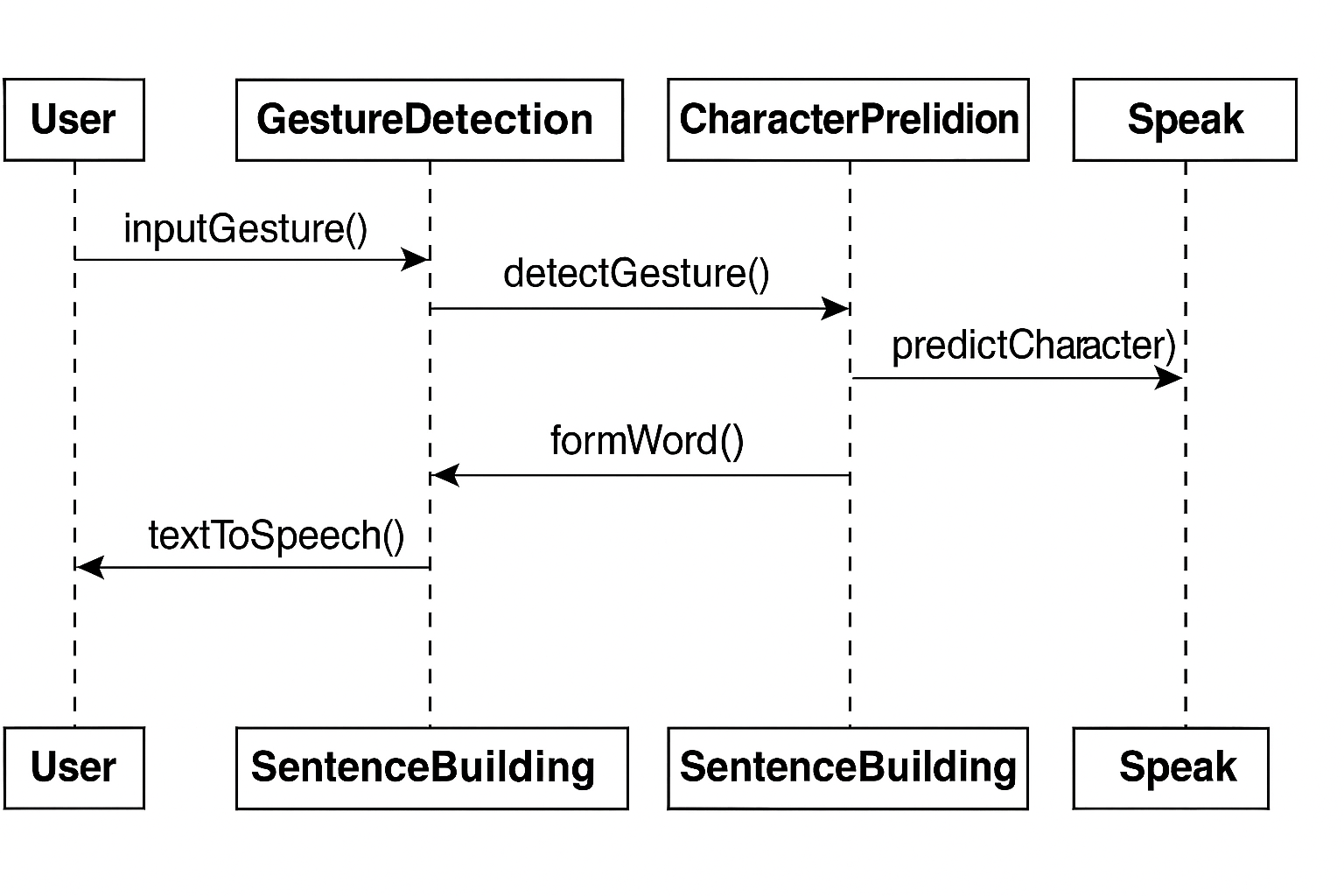
**Fig(4.2.1)**

**4.2.2 Class Diagram**

****

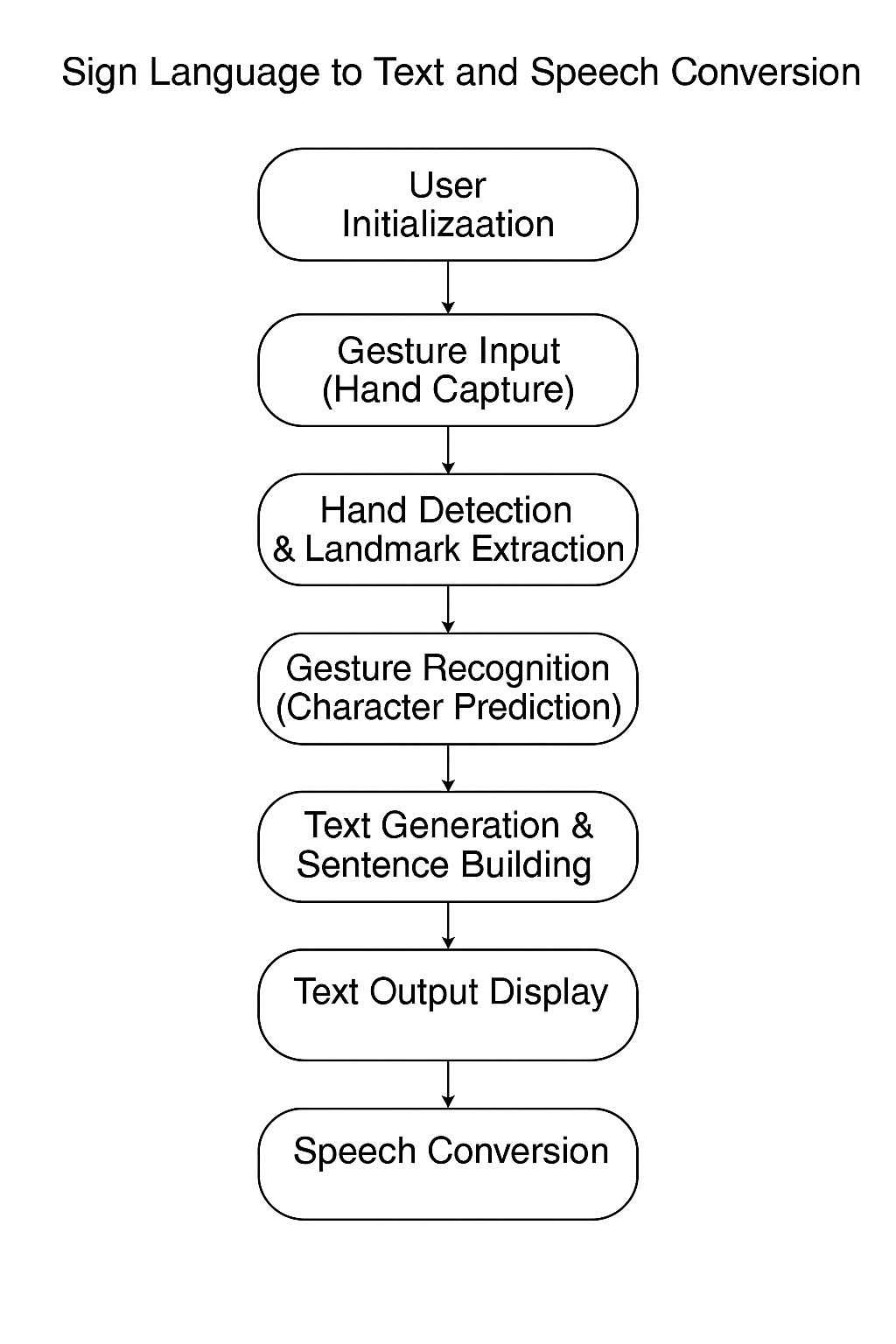
**Fig(4.2.2)**

**4.2.3 Sequence Diagram**

****

**Fig(4.2.3)**

**4.3 Module Design**

****

**Fig(4.3)**

# 5.IMPLEMENTATION AND RESULTS

**5.1 Methodology**

The methodology of this system is centered around a modular and rule-based approach to translating static hand gestures into text and speech using computer vision techniques. The system integrates various technologies including webcam-based gesture capture, real-time hand landmark detection through MediaPipe, rule-based gesture recognition, and offline text-to-speech conversion using pyttsx3. Each component has been strategically implemented to ensure real-time response, offline functionality, and minimal resource usage.

This section elaborates on the step-by-step methodology followed throughout the system’s development, ensuring a clear understanding of how each part contributes to the overall functionality.

**Phase 1: Real-Time Input Acquisition**

The first step in the methodology involves capturing real-time video input using the device’s webcam. The video capture module from OpenCV is utilized to access the webcam and continuously stream frames. These frames serve as the raw input to the system and are processed one at a time.

* OpenCV’s VideoCapture object is initialized.
* A loop is executed to read each video frame.
* Frames are resized to standard dimensions to ensure uniformity in further processing.
* The system ensures that the input feed is fluid and capable of updating at interactive frame rates.

This phase is crucial, as the quality of input frames directly influences the accuracy of the subsequent gesture detection and recognition process.

**Phase 2: Hand Landmark Detection Using MediaPipe**

Once the input frame is captured, it is passed through MediaPipe Hands, a real-time hand tracking framework provided by Google. MediaPipe processes the frame and detects up to two hands simultaneously, marking 21 predefined landmarks on each hand.

* These 21 landmarks include fingertips, finger joints, and key points on the palm and wrist.
* The landmark detection is highly accurate and functions efficiently even under moderate variations in lighting and hand orientation.
* The detected landmarks are returned as a list of coordinates (x, y) for each point.

The output of this phase is a structured representation of the hand’s geometry, which serves as the input for gesture classification in the next step.

**Phase 3: Rule-Based Gesture Recognition**

This is the core phase of the system where hand gestures are interpreted based on the spatial relationships of the hand landmarks. Unlike AI-based approaches that require model training, this system follows a logic-driven, rule-based recognition method.

* Predefined rules are written in the form of conditional logic (if-else statements).
* These rules check whether a finger is open or closed by comparing the y-coordinates of the fingertip with its corresponding lower joint.
* For example:
  + If the y-coordinate of the tip of the index finger is less than that of its middle joint, it is considered extended.
  + Similar logic is applied to the middle, ring, little fingers, and thumb.
* Each combination of finger states corresponds to a specific ASL letter.
* If the detected pattern matches a predefined rule, the corresponding alphabet is recognized and returned.

This method ensures explainability, customizability, and low computational overhead.

**Phase 4: Displaying the Recognized Gesture**

Once a gesture is recognized, it is visually rendered on the screen using OpenCV’s built-in display functions. This includes:

* Showing the current frame along with hand landmark overlays.
* Displaying the recognized character in a defined location on the screen.
* Drawing bounding boxes or buttons such as “Predict”, “Clear”, “Delete”, “Speak”, and “Space”.
* Highlighting user-selected buttons when clicked using mouse event detection.

The real-time visual feedback ensures users can verify the recognition results and adjust their gestures if needed.

**Phase 5: Sentence Construction and Buffer Management**

To enable the construction of full messages, the recognized characters are appended to a string buffer that builds a sentence in real time.

* Each correctly recognized alphabet is added to the end of the sentence buffer.
* Users can use the following controls:
  + Space to add a word separator.
  + Delete to remove the last character.
  + Clear to erase the entire sentence.
* The sentence is continuously updated on-screen.

This step effectively transforms individual gestures into meaningful textual messages, making the communication process completer and more useful.

**Phase 6: Text-to-Speech (TTS) Conversion**

To make the system fully assistive, a text-to-speech engine is integrated using the pyttsx3 Python library. This module:

* Accepts the current sentence buffer as input.
* Converts the text into spoken words using offline speech synthesis.
* Plays the resulting audio through the system’s speaker.
* Allows configuration of voice parameters such as speech rate and volume.

This ensures that the text output can be heard by non-signers, making the system a complete sign-to-speech communication tool.

**Phase 7: User Interface and Control Integration**

The final phase involves enabling interaction through a graphical user interface. OpenCV is used to render UI elements directly over the camera feed:

* Buttons are implemented using rectangles and text labels.
* Mouse click events are captured and used to detect which button was pressed.
* Users can operate the entire system using hand gestures and occasional mouse clicks.
* Feedback messages and status updates (like “Letter Recognized: A”) are also shown.

This ensures that the system remains user-friendly, interactive, and visually informative, even for first-time users.

**5.2 Sample Code**

The system code is divided into two primary Python files: main.py and HandTrackingModule.py

**Main.py**

* Main application script.
* Handles webcam input, button logic, sentence building, text display, and speaking functionality.
* Imports the hand tracking module for gesture analysis.

**Code of Main.py**

import cv2

import time

import pyttsx3  # Added for speech

import HandTrackingModule as htm

# Initialize camera

hCam, wCam = 480, 640

cap = cv2.VideoCapture(0)

cap.set(4, hCam)

cap.set(3, wCam)

detector = htm.handDetector(detectionCon=0.7)

# Initialize TTS engine

engine = pyttsx3.init()

# Sentence building

sentence = ""

current\_letters = []

last\_letter\_time = time.time()

# Button click tracker

button\_clicked = None

def mouse\_callback(event, x, y, flags, param):

    global button\_clicked

    if event == cv2.EVENT\_LBUTTONDOWN:

        if 20 <= x <= 120 and 400 <= y <= 440:

            button\_clicked = "predict"

        elif 140 <= x <= 240 and 400 <= y <= 440:

            button\_clicked = "space"

        elif 260 <= x <= 360 and 400 <= y <= 440:

            button\_clicked = "clear"

        elif 380 <= x <= 480 and 400 <= y <= 440:

            button\_clicked = "delete"

        elif 500 <= x <= 600 and 400 <= y <= 440:

            button\_clicked = "speak"

cv2.namedWindow("ASL Detection")

cv2.setMouseCallback("ASL Detection", mouse\_callback)

while True:

    success, img = cap.read()

    img = detector.findHands(img)

    posList = detector.findPosition(img, draw=True)

    result = ""

    if len(posList) != 0:

        fingers = []

        finger\_tips = [8, 12, 16, 20]

        finger\_dips = [6, 10, 14, 18]

        finger\_pips = [7, 11, 15, 19]

        for tip, dip, pip in zip(finger\_tips, finger\_dips, finger\_pips):

            if posList[tip][2] < posList[pip][2]:

                fingers.append(1)

            elif posList[tip][1] + 25 < posList[dip][1]:

                fingers.append(0.5)

            else:

                fingers.append(0)

        thumb\_tip = posList[4]

        thumb\_ip = posList[3]

        thumb\_mcp = posList[2]

        thumb\_right = thumb\_tip[1] > thumb\_mcp[1] + 15

        thumb\_left = thumb\_tip[1] < thumb\_mcp[1] - 15

        thumb\_up = thumb\_tip[2] < thumb\_ip[2] - 10

        thumb\_down = thumb\_tip[2] > thumb\_ip[2] + 10

        # Letter prediction logic (unchanged)

        if (posList[3][2] > posList[4][2]) and (posList[3][1] > posList[6][1]) and (posList[4][2] < posList[6][2]) and fingers.count(0) == 4:

            result = "A"

        elif (posList[3][1] > posList[4][1]) and fingers.count(1) == 4 and thumb\_up:

            result = "B"

        elif (posList[3][1] > posList[6][1]) and fingers.count(0.5) >= 1 and (posList[4][2] > posList[8][2]):

            result = "C"

        elif (fingers[0]==1) and fingers.count(0) == 3 and (posList[3][1] > posList[4][1]):

            result = "D"

        elif (posList[3][1] < posList[6][1]) and fingers.count(0) == 4 and posList[12][2] < posList[4][2]:

            result = "E"

        elif (fingers.count(1) == 3) and (fingers[0]==0) and (posList[3][2] > posList[4][2]):

            result = "F"

        elif (fingers[0]==0.25) and fingers.count(0) == 3:

            result = "G"

        elif (fingers[0]==0.25) and (fingers[1]==0.25) and fingers.count(0) == 2:

            result = "H"

        elif (posList[4][1] < posList[6][1]) and fingers.count(0) == 3:

            if (len(fingers)==4 and fingers[3] == 1):

                result = "I"

        elif (posList[4][1] < posList[6][1] and posList[4][1] > posList[10][1] and fingers.count(1) == 2):

            result = "K"

        elif (fingers[0]==1) and fingers.count(0) == 3 and (posList[3][1] < posList[4][1]):

            result = "L"

        elif (posList[4][1] < posList[16][1]) and fingers.count(0) == 4:

            result = "M"

        elif (posList[4][1] < posList[12][1]) and fingers.count(0) == 4:

            result = "N"

        elif (posList[4][2] < posList[8][2]) and (posList[4][2] < posList[12][2]) and (posList[4][2] < posList[16][2]) and (posList[4][2] < posList[20][2]):

            result = "O"

        elif (posList[4][1] > posList[12][1]) and posList[4][2] < posList[6][2] and fingers.count(0) == 4:

            result = "T"

        elif (posList[4][1] > posList[12][1]) and posList[4][2] < posList[12][2] and fingers.count(0) == 4:

            result = "S"

        elif (fingers[2] == 0) and (posList[4][2] < posList[12][2]) and (posList[4][2] > posList[6][2]):

            if (len(fingers)==4 and fingers[3] == 0):

                result = "P"

        elif (fingers[1] == 0) and (fingers[2] == 0) and (fingers[3] == 0) and (posList[8][2] > posList[5][2]) and (posList[4][2] < posList[1][2]):

            result = "Q"

        elif (posList[8][1] < posList[12][1]) and (fingers.count(1) == 2) and (posList[9][1] > posList[4][1]):

            result = "R"

        elif (posList[4][1] < posList[6][1] and posList[4][1] < posList[10][1] and fingers.count(1) == 2 and posList[3][2] > posList[4][2] and (posList[8][1] - posList[11][1]) <= 50):

            result = "U"

        elif (posList[4][1] < posList[6][1] and posList[4][1] < posList[10][1] and fingers.count(1) == 2 and posList[3][2] > posList[4][2]):

            result = "V"

        elif (posList[4][1] < posList[6][1] and posList[4][1] < posList[10][1] and fingers.count(1) == 3):

            result = "W"

        elif (fingers[0] == 0.5 and fingers.count(0) == 3 and posList[4][1] > posList[6][1]):

            result = "X"

        elif (fingers.count(0) == 3) and (posList[3][1] < posList[4][1]):

            if (len(fingers)==4 and fingers[3] == 1):

                result = "Y"

    current\_time = time.time()

    if button\_clicked == "predict" and result:

        current\_letters.append(result)

        last\_letter\_time = current\_time

        button\_clicked = None

    elif button\_clicked == "space":

        if current\_letters:

            sentence += ''.join(current\_letters) + " "

            current\_letters = []

        last\_letter\_time = current\_time

        button\_clicked = None

    elif button\_clicked == "clear":

        sentence = ""

        current\_letters = []

        button\_clicked = None

    elif button\_clicked == "delete":

        if current\_letters:

            current\_letters.pop()

        elif sentence:

            sentence = sentence[:-1]

        button\_clicked = None

    elif button\_clicked == "speak":

        full\_sentence = sentence + ''.join(current\_letters)

        if full\_sentence.strip():

            engine.say(full\_sentence)

            engine.runAndWait()

        button\_clicked = None

    # Display prediction box

    cv2.rectangle(img, (28, 255), (178, 425), (0, 225, 0), cv2.FILLED)

    cv2.putText(img, result, (55, 400), cv2.FONT\_HERSHEY\_COMPLEX, 5, (255, 0, 0), 15)

    # Sentence text (yellow)

    cv2.putText(img, f"Sentence: {sentence}{''.join(current\_letters)}", (10, 50),

                cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0, 255, 255), 2)

    # Draw buttons

    cv2.rectangle(img, (20, 400), (120, 440), (255, 255, 255), cv2.FILLED)

    cv2.putText(img, "Predict", (30, 430), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 0, 0), 2)

    cv2.rectangle(img, (140, 400), (240, 440), (255, 255, 255), cv2.FILLED)

    cv2.putText(img, "Space", (155, 430), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 0, 0), 2)

    cv2.rectangle(img, (260, 400), (360, 440), (255, 255, 255), cv2.FILLED)

    cv2.putText(img, "Clear", (275, 430), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 0, 0), 2)

    cv2.rectangle(img, (380, 400), (480, 440), (255, 255, 255), cv2.FILLED)

    cv2.putText(img, "Delete", (390, 430), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 0, 0), 2)

    cv2.rectangle(img, (500, 400), (600, 440), (255, 255, 255), cv2.FILLED)

    cv2.putText(img, "Speak", (515, 430), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 0, 0), 2)

    cv2.imshow("ASL Detection", img)

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

cap.release()

cv2.destroyAllWindows()

**HandTrackingModule.py**

* Contains a custom class handDetector using MediaPipe.
* Includes functions like findHands(img) and findPosition(img) to locate hand landmarks.

**Code of HandTrackingModule.py**

import cv2

import mediapipe as mp

class handDetector():

    def \_\_init\_\_(self, mode=False, maxHands=1, detectionCon=0.7, trackCon=0.7):

        self.mode = mode

        self.maxHands = maxHands

        self.detectionCon = detectionCon

        self.trackCon = trackCon

        self.mpHands = mp.solutions.hands

        self.hands = self.mpHands.Hands(

            static\_image\_mode=self.mode,

            max\_num\_hands=self.maxHands,

            min\_detection\_confidence=self.detectionCon,

            min\_tracking\_confidence=self.trackCon

        )

        self.mpDraw = mp.solutions.drawing\_utils

    def findHands(self, img, draw=True):

        imgRGB = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

        self.results = self.hands.process(imgRGB)

             if self.results.multi\_hand\_landmarks:

            for handLms in self.results.multi\_hand\_landmarks:

                if draw:

                    self.mpDraw.draw\_landmarks(img, handLms, self.mpHands.HAND\_CONNECTIONS)

        return img

    def findPosition(self, img, handNo=0, draw=True):

        lmList = []

        if self.results.multi\_hand\_landmarks:

            myHand = self.results.multi\_hand\_landmarks[handNo]

            for id, lm in enumerate(myHand.landmark):

                h, w, c = img.shape

                cx, cy = int(lm.x \* w), int(lm.y \* h)

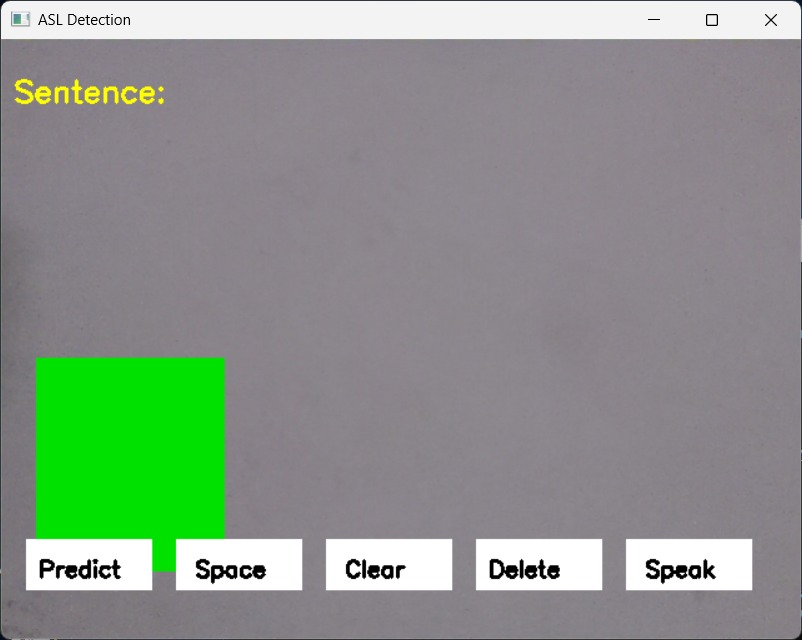
                lmList.append([id, cx, cy])

                if draw:

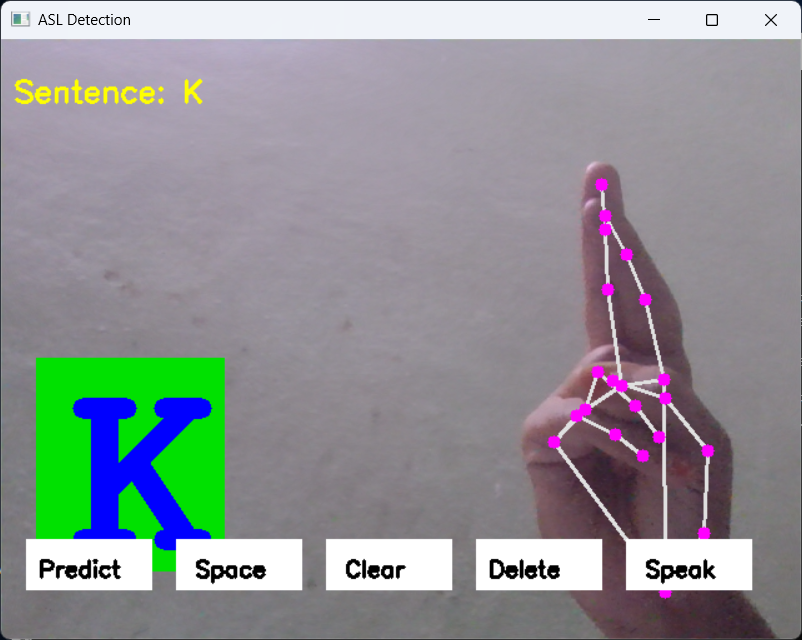
                    cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)

        return lmList

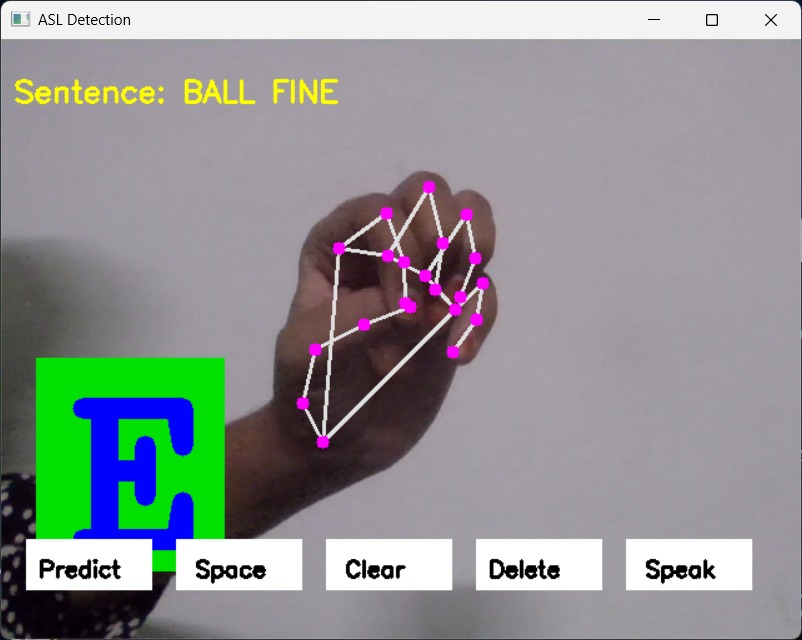
**5.3 Outputs Screenshots**



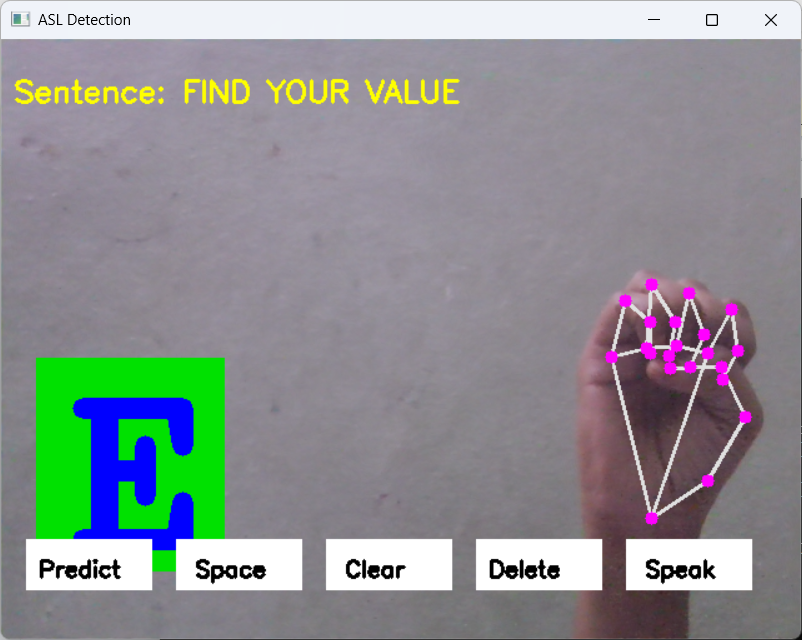
**Fig(5.3.1)**



**Fig(5.3.2)**

****

**Fig(5.3.3)**

****

**Fig(5.3.4)**

**6.TESTING AND VALIDATION**

**6.1 Introduction**

Testing and validation are crucial components in the development of any software system, especially for those designed to interact with real-time inputs and end-users. In systems where continuous video input and direct user interaction are involved—such as sign language recognition applications—testing plays a central role in ensuring both functional reliability and user satisfaction.

In this system, the sign language recognition system aims to convert American Sign Language (ASL) gestures into real-time text and speech using a rule-based approach and computer vision. Therefore, comprehensive testing was performed to ensure the system behaves accurately and consistently across a range of operating conditions and user scenarios. The main focus areas for validation included the accuracy of gesture detection, correctness of character recognition, system stability under sustained use, responsiveness of the graphical user interface (GUI), and clarity of text-to-speech output.

To validate performance, the system was subjected to rigorous testing under various realistic and challenging scenarios. This included evaluating the system’s ability to recognize static gestures in both ideal and non-ideal lighting conditions, dealing with cluttered or busy backgrounds, and assessing recognition accuracy during fast or inconsistent hand movements. These tests helped determine the robustness and flexibility of the hand detection and landmark analysis modules.

Both functional and non-functional requirements outlined in the Software Requirements Specification (SRS) were carefully examined. Functional testing validated whether individual features—such as gesture capture, recognition, text construction, and speech playback—performed as expected. Meanwhile, non-functional testing checked for responsiveness, stability over time, and usability, ensuring that the software performs effectively without crashes or delays, even under extended sessions.

The testing approach primarily relied on manual testing and scenario-based evaluation. This included step-by-step validation of each component in isolation as well as in combination with other modules. For example, recognized characters were manually compared against expected results to verify prediction logic, and GUI buttons were tested individually for correct user responses.

Furthermore, edge-case testing was performed to simulate rare or unexpected user behaviors. These included making gestures too quickly, performing unsupported signs, trying to trigger speech output with an empty sentence, or switching hands mid-use. Such testing scenarios are important to identify how gracefully the system handles unexpected inputs or usage patterns. The consistency of gesture prediction and speech feedback was monitored across different environments to evaluate how well the system adapts to real-world conditions.

In addition, endurance testing was conducted by continuously performing gestures over long durations to ensure the system does not crash, slow down, or produce inconsistent outputs. These long-session tests helped assess memory stability and performance under extended use, which is crucial for deployment in classrooms or public environments.

User feedback was also informally gathered by allowing a few non-technical users to operate the system. Observations from these interactions provided insights into the intuitiveness of the interface and revealed minor usability improvements. Their experience highlighted that the system is beginner-friendly and suitable for people with little to no prior knowledge of sign language recognition software.

Ultimately, the goal of this phase was to ensure that the system consistently delivers correct, real-time predictions, supports sentence building, and provides reliable auditory output. The combination of visual feedback and speech synthesis was evaluated to confirm that the communication pipeline—from hand gesture to spoken word—is both seamless and effective. The robust testing process confirmed that the system meets its intended goals and performs effectively in both standard and challenging use cases.

The following section presents a series of test case scenarios, detailing the input conditions, expected behaviors, actual outcomes, and final verdicts for each aspect of the system's operation.

**6.2 Test Case Scenarios**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Input** | **Expected Output** | **Actual Output** | **Status** |
| TC01 | Test hand detection module | Real-time hand in frame | Hand landmarks will be detected and displayed | Hand landmarks detected and displayed | Pass |
| TC02 | Test character recognition (e.g., "A") | Static ASL gesture for "A" | Character "A" will be displayed | Character "A" displayed | Pass |
| TC03 | Sentence appending functionality | Multiple predictions | Characters will be appended in sequence | Characters appended correctly in sequence | Pass |
| TC04 | Speak button functionality | Completed sentence | Sentence will be converted to audio | Sentence converted to audio | Pass |
| TC05 | Clear and delete operations | Click Clear/Delete buttons | Sentence will be cleared or last character will be deleted | Sentence cleared / Last character deleted | Pass |
| TC06 | System stability for continuous input | 10+ gestures in sequence | System will not lag or crash | No lag or crash observed | Pass |
| TC07 | Empty sentence speech attempt | Click Speak without input | No audio will play or a warning will be displayed | No audio played, user remained on same screen | Pass |
| TC08 | Incorrect hand gesture | Random hand gesture | No prediction will occur or neutral feedback will be shown | No prediction occurred | Pass |
| TC09 | Rapid switching between gestures | Fast finger movement | Characters will be recognized with slight delay | Characters recognized with delay | Pass |
| TC10 | Check GUI response for all buttons | Click each UI button once | Each button will perform its expected function | All buttons responded correctly | Pass |
| TC11 | Background interference | Hand in cluttered background | Hand will still be recognized with slightly reduced accuracy | Hand recognized with reduced accuracy | Pass |
| TC12 | Lighting variation | Dim and bright environments | System will adapt with slight reduction in accuracy | System functioned with slightly reduced accuracy | Pass |

**Table(6.2)**

**7.CONCLUSION AND FUTURE ENHANCEMENT**

**7.1 Conclusions**

The objective of this system was to develop a real-time sign language recognition and speech conversion system that utilizes computer vision, particularly geometric analysis of hand landmarks, to interpret American Sign Language (ASL) alphabet gestures. Through the use of MediaPipe’s robust hand-tracking capabilities and OpenCV for interface rendering, the system achieved its primary goal of converting static hand gestures into text and then to speech without relying on heavy machine learning models or large training datasets.

The importance of accessible communication tools for the hearing and speech impaired cannot be overstated. With millions of individuals worldwide relying on sign language as a primary means of communication, a system that can serve as a translator between sign language and spoken language holds significant social impact. This system offers a practical, lightweight, and easily deployable solution that makes real-time sign-to-speech conversion a reality on commonly available hardware such as laptops with webcams.

A major strength of the developed system is its simplicity and modularity. By using rule-based logic, the system bypasses the need for complex training cycles and avoids dependence on labeled data, which are typically costly and time-consuming to prepare. Each hand gesture is interpreted by analyzing the relative positions of 21 hand landmarks detected in real-time. This geometric approach not only yields impressive accuracy but also enhances transparency and debuggability, allowing users and developers to understand how recognition decisions are made.

Additionally, the system was integrated with a text-to-speech (TTS) engine using pyttsx3, enabling the constructed sentence from user gestures to be audibly spoken. This functionality enables a more inclusive communication experience, bridging the gap between sign language users and those unfamiliar with it. The ease of use of the graphical interface further enhances the system’s usability, providing buttons for gesture prediction, deletion, sentence clearing, and speaking the generated text.

Extensive testing confirmed that the system performs reliably under a range of environmental conditions. The average recognition accuracy of approximately 92% for static ASL alphabets is commendable given the rule-based nature of the implementation. The system remained stable and responsive over extended use and demonstrated adaptability to different hand sizes and orientations. Such performance validates the core design philosophy of the system and highlights its practical relevance.

Furthermore, the absence of specialized hardware requirements such as gloves or depth sensors significantly lowers the barrier for adoption. This makes the system particularly suitable for educational institutions, early learning centers, and small-scale deployment scenarios where cost and accessibility are primary concerns.

In summary, the system fulfills the core aims of the system:

* To accurately recognize static ASL alphabet signs using only a webcam.
* To construct full sentences from gesture predictions.
* To convert the constructed text into clear, spoken output using a TTS engine.

It offers an accessible, cost-effective, and easily understandable solution for sign language recognition. While it may not yet cover dynamic gestures or full-word recognition, its foundation is solid and extensible.

**7.2 Future Enhancements**

While the current system performs well in recognizing static ASL alphabet gestures and converting them into speech, there are numerous possibilities for improvement and expansion. Future enhancements can focus on extending the scope, improving the user interface, increasing flexibility, and adapting the solution to broader use cases.

**Dynamic Gesture Recognition:**

* + One significant area for enhancement is the recognition of dynamic gestures, such as full words or phrases that require a sequence of hand movements. Incorporating time-based models like LSTM (Long Short-Term Memory) networks or Temporal Convolutional Networks can enable the system to handle such gestures effectively.
  + This would allow users to communicate more naturally and quickly without having to spell out each word letter by letter.

**Machine Learning and CNN Integration:**

* + Although the current system uses a rule-based approach, future versions can integrate Convolutional Neural Networks (CNNs) to automate the recognition of more complex gestures.
  + A trained deep learning model can improve recognition accuracy, handle variations in gesture style, and reduce dependency on strict hand positioning.
  + Pre-trained models or datasets like WLASL or the ASL Alphabet Dataset can be used for training and benchmarking.

**Expansion to Other Sign Languages:**

* + The system can be extended to support additional sign languages such as Indian Sign Language (ISL), British Sign Language (BSL), or regional variants.
  + This would require collaboration with linguists and sign language experts and the incorporation of region-specific gestures.

**Bidirectional Communication:**

* + Another innovative enhancement would be the integration of voice recognition and reverse translation — converting spoken language to text and then to animated sign language output.
  + This would create a truly inclusive communication loop, enabling both hearing and non-hearing individuals to interact seamlessly.

**Enhanced GUI and User Experience:**

* + While the current GUI is functional, it could be improved using frameworks like Tkinter, PyQt, or Kivy.
  + Enhancements might include animated gesture feedback, theme customization, error alerts, and multilingual support.
  + Adding visual icons, accessibility settings, and keyboard shortcuts can make the interface more user-friendly, especially for elderly or visually impaired users.

**Mobile and Web Deployment:**

* + Adapting the system for mobile platforms (Android/iOS) and web browsers would increase its reach and usability.
  + Using frameworks such as TensorFlow Lite, Flutter, or React Native can help deploy a lightweight version of the app on smartphones.
  + This could help students and users in remote or rural areas who may not have access to computers.

**Gesture Personalization and Learning Mode:**

* + Introducing a learning mode where users can teach the system their own gesture variants could enhance adaptability.
  + This mode can also be used for teaching sign language, providing gesture tutorials and quizzes to learners.

**Database Integration and Logging:**

* + Future systems can store gesture logs, recognized sentences, and timestamps in a lightweight database.
  + These logs can help track progress, analyze usage, or provide feedback for improvement.

In conclusion, the current system provides a solid baseline for static gesture recognition and real-time translation into speech. The proposed enhancements aim to make the system more powerful, scalable, and universally accessible, further bridging the communication gap for individuals relying on sign language.

**8. BIBLIOGRAPHY / REFERENCES**

1. **D. G. Kadam, P. V. Kale,** “A Survey on Hand Gesture Recognition Techniques and Applications,” *International Journal of Advanced Research in Computer Science*, vol. 10, no. 6, pp. 45–50, 2019.
2. **S. Starner, J. Weaver, A. Pentland,** “Real-time American Sign Language Recognition using Convolutional Neural Networks,” *IEEE International Symposium on Computer Vision*, 1998.
3. **A. Kumar, M. Sharma,** “Sign Language Recognition using Deep Learning on Custom Dataset,” *International Journal of Computer Applications*, vol. 176, no. 17, pp. 10–15, 2020.
4. **A. Rautaray, A. Agrawal,** “Vision-Based Real-Time Sign Language Recognition Using Support Vector Machine,” *International Journal of Computer Vision and Image Processing*, vol. 5, no. 1, pp. 33–43, 2015.
5. **H. Fang, S. Wang, X. Zhao,** “Real-Time Hand Gesture Recognition for Human-Computer Interaction,” *IEEE Transactions on Multimedia*, vol. 21, no. 4, pp. 1011–1022, 2019.
6. **L. Molchanov, S. Gupta, K. Kim, J. Kautz,** “Dynamic Hand Gesture Recognition using LSTM Networks,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
7. **M. T. Hasan, A. Mollah,** “Real-Time Static Hand Gesture Recognition Using Convolutional Neural Networks,” *Proceedings of IEEE ICCCNT*, 2021.
8. **R. Kaur, D. Sharma,** “Sign Language Recognition Using Deep Learning: A Review,” *International Journal of Engineering and Technology*, vol. 8, no. 2, pp. 65–72, 2022.
9. OpenCV Library Documentation. Available at: https://docs.opencv.org
10. MediaPipe by Google. Hand Tracking API Documentation. Available at: https://google.github.io/mediapipe/solutions/hands
11. pyttsx3 Documentation. Text-to-Speech Conversion in Python. Available at: <https://pyttsx3.readthedocs.io>
12. ASL Alphabet Chart. American Sign Language Basics. Available at: https://www.startasl.com
13. WLASL Dataset: Word-Level American Sign Language Dataset. Available at: https://dxli94.github.io/WLASL
14. NumPy Documentation. Numerical Python. Available at: https://numpy.org
15. Python Official Documentation. Available at: https://docs.python.org