

**Tribhuvan University**

**Faculty of Humanities and Social Studies**

lkjasd

**(Python based project)**

**A PROJECT REPORT**

**Submitted to:**

**Department of Computer Application**

**Southwestern state College**

***In the partial fulfillment of the requirements for the bachelors in Computer***

***Application***

Submitted by:

Chiran Raj Thapa (6-2-530-9-2020)

July 2024 A.D

**Under the supervision of**

**Mr. Kiran Ghimire**



**Tribhuvan University**

**Faculty of Humanities and Social Science**

**Southwestern State College**

# Supervisor’s Recommendation

I hereby recommend that this project is prepared under my supervision by **Chiran Raj Thapa** entitled “**Sign language Detection** (python based project)” in the partial fulfillment for the degree of Bachelor of Computer Application is recommended for the final evaluation.

**Mr.Kiran Ghimire**

**Supervisor**

**Southwestern State college**



**Tribhuvan University**

**Faculty of Humanities and Social Science**

**College**

# LETTER OF APPROVAL

This is to certify that this project is prepared by **Chiran Raj Thapa** entitled “**Sign Language Detection (python based project)**” in the partial fulfillment for the degree of Bachelor of Computer Application has been evaluated. In our opinion it is satisfactory in the scope and quality as a project for the required degree.

**Kiran Ghimire** **Mr. Bijay Babu Regmi Project Supervisor Internal Supervisor BCA Department SWSC**

**Bikash Parajuli External Examiner**

**Internal Supervisor Tribhuvan University**

# ABSTRACT

The **Sign Language Detection System** is a tool for recognizing static hand gestures using image processing techniques. It leverages the **OpenCV library** for real-time video capture and processing, focusing on detecting hand shapes and movements. By using methods like **contour detection**, **convex hulls**, and **defects**, the system identifies and isolates the hand region for accurate gesture recognition.

This system can recognize various gestures and can be used in applications such as **human-computer interaction**, **virtual controls**, and basic **sign language recognition**. Its lightweight design makes it efficient and suitable for real-time applications. Additionally, it serves as a foundation for advanced features like **gesture-to-action mapping** and integration into **gesture-based control systems**.

The system can help bridge communication gaps, offering a practical solution for interpreting basic sign language. Future improvements may include **dynamic gesture recognition**, **multi-hand detection**, or support for larger sign language vocabulary. Its simplicity and adaptability make it a valuable tool for research and real-world applications.

*Keyword: Python, OpenCV*

# ACKNOWLEDGEMENT

For the partial fulfillment of this semester's project, I would like to express my heartfelt gratitude to everyone who directly or indirectly contributed to its successful completion. There were moments when I felt this project was too challenging and feared I might not complete it on time. However, I am deeply thankful to everyone who provided encouragement, guidance, and support, which motivated me to see it through.

I am especially thankful to our supervisor, **Kiran Ghimira**, and our coordinator, **Ravi Chapagain**, for their invaluable support and assistance whenever I faced difficulties during the project. Their guidance made a significant difference and helped me overcome various challenges.

I also extend my sincere thanks to my classmates, who continuously inspired me to stay committed and reminded me of the competitive spirit throughout. Their encouragement kept me motivated to push forward.

A special note of gratitude goes to all our professors for their guidance and support, which played a vital role in shaping the project's direction and outcomes.

Lastly, I am grateful to **Tribhuvan University** for providing this opportunity through the course of **Computer Applications**. This project has not only helped me understand project ethics at an early stage but also allowed me to enhance my skills and knowledge. Thank you all for making this journey a rewarding experience.

**With Respect**

**Chiran Raj Thapa**

**(6-2-530-9-2020)**

**TABLE OF CONTENTS**

[Supervisor’s Recommendation ii](#_Toc183296335)

[LETTER OF APPROVAL iii](#_Toc183296336)

[ABSTRACT iv](#_Toc183296337)

[ACKNOWLEDGEMENT v](#_Toc183296338)

[LIST OF TABLES x](#_Toc183296339)

[LIST OF FIGURES xi](#_Toc183296340)

[CHAPTER 1 : INTRODUCTION 1](#_Toc183296341)

[1.1 Introduction 1](#_Toc183296342)

[1.2 Problem statement 2](#_Toc183296343)

[1.3 Objective 2](#_Toc183296344)

[1.4 Scope and Limitations 3](#_Toc183296345)

[1.4.1 Scope 3](#_Toc183296346)

[1.4.2 Limitations 3](#_Toc183296347)

[1.5 Development Methodology 3](#_Toc183296348)

[1.6 Report Organization 6](#_Toc183296349)

[CHAPTER 2 BACKGROUND STUDY AND LITERATURE REVIEW 7](#_Toc183296350)

[2.1 Background Study 7](#_Toc183296351)

[2.2 Literature review 7](#_Toc183296352)

[2.2.1 Existing system 8](#_Toc183296353)

[CHAPTER 3 SYSTEM ANALYSIS AND DESIGN 10](#_Toc183296354)

[3.1 System Analysis 10](#_Toc183296355)

[3.1.1 Requirement Analysis 10](#_Toc183296356)

[3.1.1.1 Functional requirement 10](#_Toc183296357)

[3.1.1.1 Non-functional requirement 11](#_Toc183296358)

[3.1.2 Feasibility analysis 11](#_Toc183296359)

[3.1.2.1 Technical Feasibility 11](#_Toc183296360)

[3.1.2.2 Operational Feasibility 11](#_Toc183296361)

[3.1.2.3 Economic Feasibility 12](#_Toc183296362)

[3.1.2.4 Schedule Feasibility 12](#_Toc183296363)

[3.1.3 Object Modeling: Object and class diagram 13](#_Toc183296364)

[3.1.4 Dynamic Modeling using state and sequence diagram 14](#_Toc183296365)

[3.1.5 Process Modeling using Activity Diagram 17](#_Toc183296366)

[3.2 System Design 18](#_Toc183296367)

[3.2.1.1 Refinement of Sequence, and activity diagram 19](#_Toc183296368)

[3.2.2 Component Diagram 19](#_Toc183296369)

[3.2.3 Deployment Diagram 19](#_Toc183296370)

[3.3 Algorithm details 20](#_Toc183296371)

[3.3.1 Keypoint Classifier 20](#_Toc183296372)

[3.3.2CNN 21](#_Toc183296373)

[CHAPTER 4 IMPLEMENTATION AND TESTING 25](#_Toc183296374)

[4.1 Implementation 25](#_Toc183296375)

[4.1.1 Tools Used (Programming Languages) 25](#_Toc183296376)

[4.1.1.1 Implementation Details of Modules (Description of Procedures/ functions) 26](#_Toc183296377)

[4.2 Testing 27](#_Toc183296378)

[4.2.1 Test Case for Unit Testing 27](#_Toc183296379)

[4.2.2 Test Case for System Testing 30](#_Toc183296380)

[CHAPTER 5 CONCLUSION AND FUTURE RECOMMENDATIONS 32](#_Toc183296381)

[5.1 Lesson learnt/ Outcome 32](#_Toc183296382)

[5.2 Conclusion 32](#_Toc183296383)

[5.3 Future Recommendation 33](#_Toc183296384)

[REFERENCES 34](#_Toc183296385)

[APPENDIX 35](#_Toc183296386)

**LIST OF ABBREVIATIONS**

**ML**: Machine Learning

**CNN**: Convolutional Neural Network

**RGB**: Red, Green, Blue (image color space)

**HSV**: Hue, Saturation, Value (color model)

**ACC**: Accuracy

**CV**: OpenCV

**DA**: Data Augmentation

# LIST OF TABLES

[Table 1:Gesture Detection with valid Gestures 28](#_Toc183296135)

[Table 2: Gesture Detection with Invalid Gestures 30](#_Toc183296136)

# LIST OF FIGURES

[Figure 1:Use case Diagram 10](#_Toc183296138)

[Figure 2:Ghantt Chart 12](#_Toc183296139)

[Figure 3:Class Diagram 13](#_Toc183296140)

[Figure 4:Sequence Diagram 15](#_Toc183296141)

[Figure 5:State Diagram for user 16](#_Toc183296142)

[Figure 6:Activity Diagram 17](#_Toc183296143)

[Figure 7:System Architecture 18](#_Toc183296144)

[Figure 8:Component Diagram 19](#_Toc183296145)

[Figure 9:Deployment Diagram 20](#_Toc183296146)

# CHAPTER 1 : INTRODUCTION

## 1.1 Introduction

This project presents a Python-based **sign language detection system** designed to improve human-computer interaction by identifying hand gestures in real time. The system uses the **OpenCV library** for computer vision and image processing, enabling it to detect and recognize hand movements captured through a live video feed or camera. By applying techniques such as **contour detection**, **convex hull formation**, and **defect point analysis**, the system identifies the shape and structure of the hand to interpret gestures.

The current version focuses on detecting gestures like an open palm, fist, or specific finger positions, laying a strong foundation for future extensions. The system works by detecting the hand in a video frame, extracting its features, and analyzing its geometry to identify gestures.

It is lightweight and efficient, designed to run smoothly on standard hardware with minimal resource requirements. This project highlights how computer vision and basic image processing techniques can create intuitive and user-friendly interfaces.

Future upgrades may include the use of machine learning for more accurate detection, multi-hand tracking, and custom gesture libraries for specific applications. With the growing need for touchless systems, this **sign language detection project** takes an important step toward creating seamless and immersive user experiences.

## 1.2 Problem statement

The problem statement for **Sign Language Detection System** is that current methods of communication for individuals who rely on sign language, such as human interpreters or written communication, are often inefficient and not always readily available. This limitation leads to significant delays in interaction, making real-time communication challenging and frustrating, especially in critical settings like healthcare, education, and public services.

Traditional solutions like hiring interpreters can be costly and impractical in many situations, while written communication may not fully capture the nuances of sign language. Furthermore, these methods often fail to provide an inclusive and seamless experience, leaving individuals feeling isolated or misunderstood.

Handling sign language interpretation through manual or conventional means requires substantial resources and may still result in gaps in understanding. This highlights the need for an accessible, real-time solution to bridge the communication gap and promote inclusivity for individuals relying on sign language.

## 1.3 Objective

* Ensure accurate recognition of basic gestures with minimal resource usageTo offer an accurate, efficient, and user-friendly interface that can adapt to various dynamic environments.
* Design the system to support future expansions, including more gestures and custom libraries.
* Enable integration into real-world applications like virtual environments, healthcare, and assistive technologies

## 1.4 Scope and Limitations

### 1.4.1 Scope

* The system will focus on recognizing basic hand sign gestures, such as open palms, fists, and specific finger configurations, to interpret sign language.
* The system will operate in real-time, processing live video feeds to detect and interpret gestures with minimal delay
* The system can be integrated into existing platforms and devices, such as mobile apps, smart TVs, and virtual environments, to promote inclusivity and better communication.

### 1.4.2 Limitations

* The system may only recognize a limited set of basic gestures and may not be able to handle complex or regional variations of sign language.
* Lighting conditions, background noise, or camera quality could affect the accuracy and performance of gesture recognition.
* The system’s performance may be impacted by hardware limitations, especially on devices with lower processing power

## 1.5 Development Methodology

For the **Sign Language Detection System**, the **Agile development methodology** would be an effective approach due to its flexibility, iterative progress, and continuous feedback, which is crucial for improving the system as new gestures or features are added. Below is an outline of how the development process can be structured:

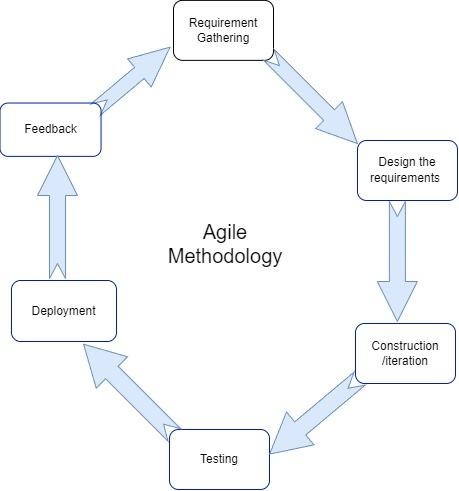


Figure 1.1 Agile Model of Hand Gesture Recognition

1. Requirement Gathering & Analysis:
   * Initial discussions with stakeholders (e.g., users, domain experts) to identify the key features and gestures to be recognized (e.g., open palm, fist, specific finger configurations).
   * Defining the scope, such as real-time processing, camera integration, and accessibility for users with physical impairments.
2. Planning & Iteration Setup:
   * Break down the project into smaller, manageable iterations or sprints, each with a specific goal (e.g., one sprint could focus on gesture recognition, another on improving accuracy or handling different lighting conditions).
   * Set clear, achievable milestones for each iteration to maintain progress.
3. Design:
   * Design the system architecture (e.g., integrating OpenCV for gesture recognition, setting up the camera feed, defining data flow).
   * Design the user interface (if applicable) to ensure that the system is accessible and easy to use.
4. Implementation (Iterative Process):
   * Start developing the core features in small iterations. Each sprint may involve:
     + Gesture detection and feature extraction.
     + Real-time processing with continuous feedback loops.
     + Initial testing on various environments and conditions.
   * After each iteration, deploy the updates, review the results, and refine the system based on user feedback or testing.
5. Testing & Quality Assurance:
   * Perform rigorous testing after each iteration to ensure the accuracy of gesture recognition, speed, and overall functionality.
   * User testing to ensure the system’s accessibility and efficiency in real-world conditions (e.g., varied lighting, different hand shapes).
6. Deployment & Integration:
   * Deploy the system in a real environment and integrate it with applications where it can be used (e.g., virtual environments, assistive technologies).
7. Review & Feedback:
   * At the end of each iteration, gather feedback from users, stakeholders, and developers to identify areas for improvement.
   * Use the feedback to plan for new features or enhancements in the next iteration.
8. Maintenance & Updates:
   * Continuously update the system to improve its functionality, recognize additional gestures, and handle new use cases.
   * Address any bugs or issues that arise and release regular updates to ensure smooth operation.

By following the Agile methodology, the development process remains flexible and adaptive, allowing for continuous improvement and the ability to respond to evolving requirements or challenges throughout the development cycle of the Sign Language Detection System

## 1.6 Report Organization

**Chapter 1**, which introduces the **Sign Language Detection System**, outlining its objectives, the problem statement, report structure, project scope, limitations, and the methodology used for the development process.

**Chapter 2** presents the background study, explaining the need for the project in the context of modern-day challenges and follows with a review of related literature and similar projects.

**Chapter 3** covers data modeling and process modeling techniques, presenting the system's requirements and feasibility study. It includes designs such as Use Case Diagrams, Gantt Chart, Entity Relationship Diagram (ERD), Process Modeling (Level 0 and Level 1 DFD), System Architecture, Database Schema Design, Interface Design, and Physical DFD.

**Chapter 4** discusses the implementation and testing of the project. This chapter provides an overview of the tools used, explains the testing methods, and outlines the modules developed for the system.

**Chapter 5** concludes the report by summarizing the accomplishments of the project, presenting its findings, and suggesting recommendations for future enhancements. It provides a reflection on the project's objectives, scope, and limitations while emphasizing the value and potential for further development.

# CHAPTER 2 BACKGROUND STUDY AND LITERATURE REVIEW

## 2.1 Background Study

The planned **Sing Language Detection application** is designed to improve user interaction by offering an intuitive, touchless control system that enhances accessibility and engagement. The system focuses on recognizing and interpreting hand sign gestures in real time, allowing users to interact with technology in a more natural and effective way. Using computer vision techniques, the application identifies various hand gestures, providing a seamless experience and reducing reliance on traditional input devices.

In **Nepal**, there is a growing need for accessible technology to support individuals with physical impairments or those who prefer alternative methods of interaction. Currently, there are few systems that cater to these needs, and awareness of gesture recognition technology remains limited. This application not only aims to enhance the user experience but also serves as an educational tool to raise awareness about the potential of gesture recognition in Nepal.

The application is designed for use by everyone, regardless of their physical abilities, and demonstrates how innovative solutions can improve accessibility and interaction. It aims to ensure that all individuals can engage with technology in a meaningful and inclusive way

## 2.2 Literature review

**Sign language detection** has gained significant attention due to its potential to improve human-computer interaction (HCI) across various fields such as assistive technologies, virtual communication, education, and accessibility for individuals with hearing impairments.

The field has advanced considerably with the help of tools like **Python** and **OpenCV**. Ray et al. explored this domain in their study, *"Sign Language Detection using Python,"* published in the *International Journal on Future Revolution in Computer Science & Communication Engineering*. They demonstrated how Python can be effectively used to build robust sign language detection systems, highlighting its simplicity and suitability for real-world applications【1】.

Harini et al. further expanded on these ideas in their work, *"Sign Language Detection using OpenCV and Python,"* published in *Springer eBooks*. They emphasized the integration of OpenCV with Python to achieve real-time sign language detection, which enhances system accuracy and robustness under diverse conditions【2】.

Ismail et al., in their paper, *"Sign Language Detection on Python and OpenCV,"* published in the *IOP Conference Series: Materials Science and Engineering*, focused on real-time sign language detection using color segmentation and feature extraction techniques, advancing the precision of gesture recognition for communication【3】.

In another study, Baihaqi et al. in *"Real-Time Sign Language Detection for Humanoid Robot Control using Python CVZone,"* published in *Lecture Notes in Networks and Systems*, demonstrated how sign language can be used for controlling humanoid robots. Their research highlighted the **CVZone** library in Python, showcasing its potential for real-time human-robot interaction and the future of sign language-based systems in robotics【4】.

### 2.2.1 Existing system

**Microsoft's Kinect Sign Language Recognition System**: Microsoft has developed a **sign language recognition system** using the **Kinect** sensor, which uses depth-sensing technology to track hand movements and interpret sign language gestures. This system allows users to communicate using American Sign Language (ASL) through a Kinect device, which can detect the position and movement of the user's hands and arms. The system identifies gestures and converts them into text or speech. Although this system is capable of recognizing a range of sign language gestures, it requires a specialized hardware setup (Kinect sensor) and is typically used for research or specific applications rather than general public use. **SignAll**: **SignAll** is an automated sign language translation system that uses machine learning to recognize and translate American Sign Language into text. It uses a combination of computer vision, deep learning, and natural language processing techniques to identify hand gestures, facial expressions, and body movements. The system uses multiple cameras to capture the user's signs from different angles and provides real-time translation into text or speech. This system is primarily aimed at improving communication between deaf individuals and those who do not know sign language. However, it requires multiple cameras and a controlled environment to work effectively, limiting its accessibility and portability.

**DeepASL**: **DeepASL** is a deep learning-based system designed to recognize American Sign Language (ASL) gestures from video input. The system uses a Convolutional Neural Network (CNN) to process the images or video frames and identify gestures. DeepASL is trained on a large dataset of ASL signs and can recognize common signs with high accuracy. It is capable of performing real-time recognition and provides feedback by converting signs into text. While the system performs well for simple gestures, its accuracy may decrease for more complex signs, and it requires high-quality video input to function optimally.

**Sign Language Recognition Using OpenCV and Python**: A popular open-source approach is the use of **OpenCV** and **Python** for sign language recognition. Developers use techniques such as contour detection, hand tracking, and feature extraction to identify gestures. With **OpenCV**, real-time recognition can be achieved using standard webcams, making it more affordable and accessible compared to systems requiring specialized hardware. While these systems can detect simple signs and gestures, they often struggle with more complex sign language structures that involve facial expressions and non-manual signals (NMS), which are essential in many sign languages like ASL.

# CHAPTER 3 SYSTEM ANALYSIS AND DESIGN

## 3.1 System Analysis

.It is a problem-solving activity that requires intensive communication between the system users and system developers. It is an important phase of any system development process.The project will be explained using dataflow diagrams, flowcharts, use-case diagrams, connection and entity diagrams, and so on

### 3.1.1 Requirement Analysis

The requirement analysis for the **Sign Language Detection System** involves identifying and defining the necessary features, functionalities, and requirements for the system to function effectively and efficiently.

#### 3.1.1.1 Functional requirement

* The system shall detect and recognize hand gestures in real-time.
* The system shall allow users to calibrate the camera and adjust settings for optimal performance.
* The system shall track and update the hand’s position continuously during gesture recognition.

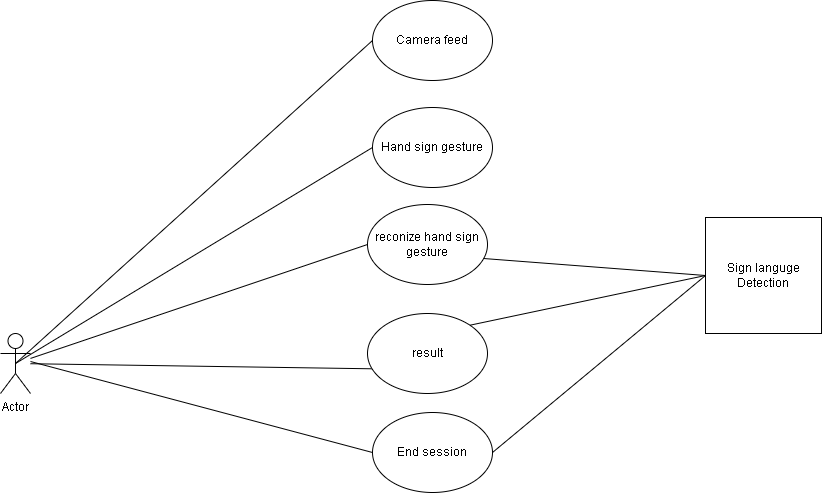


Figure :Use case Diagram

#### 3.1.1.1 Non-functional requirement

* The system must have a simple and user-friendly interface that is easy for all users to navigate.
* The system must always be available for use, ensuring continuous accessibility.
* The system must provide fast and responsive performance with minimal delay in gesture recognition.
* The system must be maintainable with clear documentation, allowing for easy updates and improvements.
* The system must be scalable to accommodate various sign languages and user preferences.
* The system must be reliable, with minimal downtime or errors during use.
* The system must be compatible across different devices and operating systems (e.g., Windows, macOS, Android, iOS).

### 3.1.2 Feasibility analysis

A feasibility study for the **Sign Language Detection System** involves evaluating whether the development and deployment of the system are practical and beneficial.

#### 3.1.2.1 Technical Feasibility

The system was developed and tested on an Intel Core i5 processor with 8GB RAM, running on Windows, macOS, and Linux platforms. The system is compatible with standard webcams, ensuring it can run effectively on most modern devices, making it accessible to a wide range of users with different hardware configurations.

#### 3.1.2.2 Operational Feasibility

The system must be easy for users to operate, requiring minimal setup and intervention. It should accurately recognize and interpret hand gestures in real-time, providing clear text or audio feedback. The system must also be scalable to handle multiple users and different sign language gestures. It should work in varying lighting conditions and with common camera setups, ensuring its practicality for diverse environments.

#### 3.1.2.3 Economic Feasibility

The project is highly cost-effective as it uses open-source technologies like Python, OpenCV, and TensorFlow, which are free and widely available. There are no significant costs associated with proprietary software, and the hardware requirements are minimal, making the system affordable to develop and deploy. The benefits of improving accessibility through gesture recognition outweigh the development costs, confirming the project’s economic feasibility.

#### 3.1.2.4 Schedule Feasibility

A graph with orange bars

Description automatically generated with medium confidence

Figure :Ghantt Chart

### 3.1.3 Object Modeling: Object and class diagram

A **class diagram** in UML is a static structure diagram that visually represents the architecture of a system. It shows the system's **classes**, each with its attributes (properties) and **operations** (methods). These classes are connected by various relationships, such as associations, inheritances, or dependencies, which indicate how the objects of one class interact with the objects of another class. The class diagram provides an organized overview of the system's components and their connections, making it an essential tool for understanding the structure and design of the system. It helps developers and designers map out the system's functionality, ensuring a clear blueprint for development and implementation

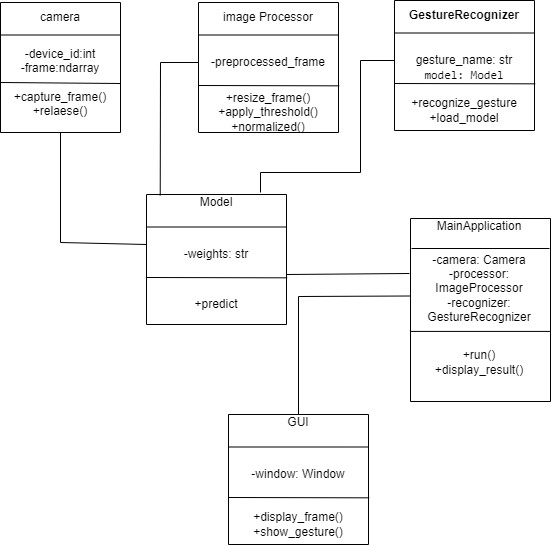


Figure :Class Diagram

In Figure 3.3, the diagram illustrates the various classes in the sing language detection system and their relationships. The primary classes include GestureRecognizer, CameraFeed, GestureModel, and UserInterface. The GestureRecognizer class is associated with the HandTracking class, which has a composition relationship with the GestureData class and an aggregation relationship with the CameraFeed class. Similarly, the GestureHistory class maintains an aggregation relationship with the GestureData class, storing previously recognized gestures for future reference.

### 3.1.4 Dynamic Modeling using state and sequence diagram

A **sequence diagram** in UML is used to depict the flow of messages exchanged between different objects during a specific interaction, helping to visualize the sequence of events over time. In the context of the **Sign Language Detection System**, the diagram illustrates how the user interacts with the system. The **User** (actor) begins by performing a sign language gesture. The **CameraFeed** object captures these hand movements in real-time through the webcam. The **GestureRecognizer** then processes the captured frames, recognizing the gesture being made. Once the gesture is identified, the **GestureRecognizer** sends the recognized gesture information to the **UserInterface**, where it is displayed as either text or audio feedback. Throughout this process, the system ensures seamless communication between the user, camera, gesture recognition module, and user interface, typically managed on the **Server** or local device, facilitating efficient real-time gesture recognition.

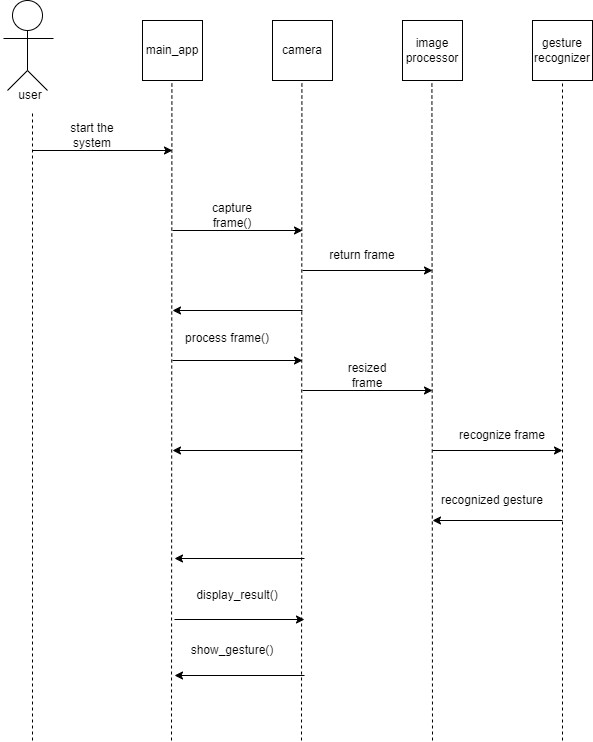


Figure :Sequence Diagram

A state machine tracks the current state of an object and can transition between states or trigger actions based on inputs. It represents the various conditions an object can hold.

In the state diagram for the hand gesture recognition system, the user starts on the main interface. If they activate the gesture recognition feature, the system transitions to the camera state, where hand tracking begins. Alternatively, if the user selects a predefined gesture, the system processes this input and executes the corresponding action.

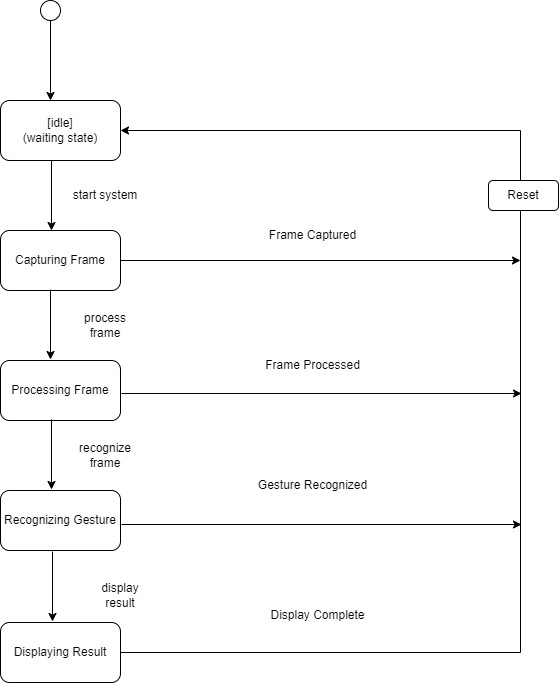


Figure :State Diagram for user

### 3.1.5 Process Modeling using Activity Diagram

In the diagram below, the user starts at the **command line interface** of the **sign language detection system**. From here, they can either choose to start **sign language detection** or view the instructions. If the user opts to begin **sign language detection**, the system captures hand movements in real time, processes them, and identifies the specific gestures related to sign language. Once a gesture is recognized, the system displays the corresponding name or description of the action. This approach ensures a seamless, real-time interaction, providing an intuitive and user-friendly experience with the **sign language detection system**.

A diagram of a process

Description automatically generated

Figure :Activity Diagram

## 3.2 System Design

**System design** refers to the process of defining the architecture, interfaces, and components that make up a system. It involves applying system theory principles to ensure that the system is effective, efficient, and capable of meeting its objectives.

**System Architecture** provides the structural framework for the system, detailing how different components interact and relate to each other. In the context of a **sign language detection system**, the architecture integrates key elements such as **detection**, **tracking**, and **recognition**. The system is designed to work efficiently across multiple platforms, allowing for flexibility and wide accessibility. The user initiates the application, which triggers a real-time processing pipeline capable of detecting and interpreting hand movements. This multi-layered architecture ensures smooth communication and interaction between the system's various components, optimizing performance and user experience.

By organizing the system into modular components, the architecture ensures that each part (e.g., gesture detection, data processing, and feature display) operates cohesively, contributing to an effective sign language detection solution.

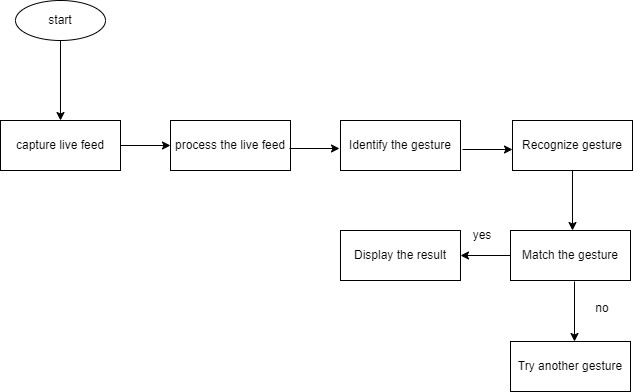


Figure :System Architecture

#### 3.2.1.1 Refinement of Sequence, and activity diagram

The refinement of sequence, and activity are not done as there were not big changes made in those diagrams. The system was designed on the basis of the previously created sequence and activity diagram in analysis phase.

#### 3.2.2 Component Diagram

A **UML component diagram** shows the structure of a system by dividing it into separate building blocks called **components.** Each component represents a logical piece of the system, like a function or service. The diagram focuses on how these components interact, share information, and depend on one another. The diagram also shows which pieces are connected and how they collaborate to achieve specific tasks.

It’s used to ensure that all parts of the system fit well together and work efficiently.

A diagram of a computer system

Description automatically generated

Figure :Component Diagram

#### 3.2.3 Deployment Diagram

A **UML component diagram** visually represents the structure of a system by breaking it down into discrete components. Each component represents a logical part of the system, such as a specific function or service. The diagram highlights the interactions between these components, showing how they exchange information, rely on each other, and work together to perform the system's tasks.

In the case of a **sign language detection system**, the components could include modules for **gesture detection**, **image processing**, **gesture recognition**, and **user interface**. The component diagram outlines how each of these parts communicates and collaborates to ensure the system works as a whole. By mapping out these relationships, the diagram helps to ensure that all components integrate seamlessly and function efficiently, providing clarity on the overall architecture and the flow of data within the system.

A diagram of a computer system

Description automatically generated

Figure :Deployment Diagram

## 3.3 Algorithm details

### 3.3.1 Keypoint Classifier

A **KeyPoint Classifier** is a machine learning model designed to identify specific patterns or gestures using the coordinates of key points (landmarks) detected in images or video frames. For **sign language detection**, these key points correspond to specific locations on the hand, such as finger tips, joints, and the palm. The classifier processes this data to recognize gestures accurately.

**Steps Involved in Using a KeyPoint Classifier:**

1. **Training Data**:  
   The classifier is trained on labeled datasets where each set of landmarks corresponds to a specific sign language gesture. These datasets serve as the foundation for teaching the model how to interpret gestures.
2. **Feature Extraction**:  
   When a hand gesture is detected, the system extracts key points or landmarks from the hand. The extracted data is pre-processed by normalizing the coordinates to a standard scale, centering them around a reference point (e.g., the wrist), and converting them into a one-dimensional format.
3. **Classification**:  
   The processed landmarks are fed into the classifier, which uses algorithms to determine the gesture. The classifier assigns a **gesture ID** or label that corresponds to the detected gesture.
4. **Output**:  
   The classifier's output is typically an integer ID or a textual label representing the recognized sign language gesture. This result can be displayed or used for further actions in the application.

By focusing on key points like finger tips, joints, and palm positions, the KeyPoint Classifier provides an efficient and accurate way to interpret gestures for real-time **sign language detection**.

### 3.3.2CNN

A **Convolutional Neural Network (CNN)** is a specialized neural network architecture designed for processing structured grid data, such as images. In the context of **sign language detection**, CNNs learn spatial hierarchies of features (from simple edges to complex shapes) to classify and recognize gestures captured in image or video frames. Below is a detailed explanation of the CNN process, including parameters, conditions, and how it works.

**Detailed Process of CNN for Sign Language Detection**

**1. Input Image Pre-processing**

* **Input Data**: Images of hand gestures (e.g., 128x128 pixels RGB images).
* **Normalization**: Pixel values are scaled to a range of 0–1 to improve numerical stability during training.
* **Data Augmentation**: Techniques like rotation, flipping, cropping, and brightness adjustments are applied to increase the diversity of the dataset.
* **Condition**: Ensure consistent image size and proper labeling for gestures (e.g., "Hello," "Thank you," "Yes," "No").

**2. CNN Layers**

**a. Convolutional Layers**

* **Function**: Extract spatial features from images.
* **Parameters**:
  + **Filter/Kernel Size**: Common sizes are 3×33 \times 33×3 or 5×55 \times 55×5. Filters slide over the image to detect features.
  + **Stride**: Determines the step size for moving the filter (e.g., stride = 1 for detailed features).
  + **Padding**: Adds borders to images to preserve their size after convolution (e.g., "same" padding maintains input size).
  + **Number of Filters**: Typically starts small (e.g., 32 filters) and increases with deeper layers (e.g., 64, 128 filters).
* **Output**: Feature maps highlighting edges, corners, and textures in the image.

**b. Activation Function**

* **Function**: Introduces non-linearity to the network, allowing it to learn complex patterns.
* **Common Choice**: ReLU (Rectified Linear Unit) ReLU(x)=max⁡(0,x)\text{ReLU}(x) = \max(0, x)ReLU(x)=max(0,x).
* **Condition**: Eliminates negative pixel values for computational efficiency.

**c. Pooling Layers**

* **Function**: Reduce spatial dimensions (downsampling) while retaining essential features.
* **Types**:
  + **Max Pooling**: Retains the maximum value in each region (common for image data).
  + **Average Pooling**: Calculates the average of values in each region.
* **Parameters**:
  + **Pool Size**: Typical sizes are 2×22 \times 22×2.
  + **Stride**: Determines the step size for pooling (usually 2).
* **Output**: Smaller feature maps that focus on critical features, reducing computation.

**d. Flattening**

* Converts the 2D feature maps into a 1D vector for input into fully connected layers.

**e. Fully Connected Layers**

* **Function**: Perform classification based on extracted features.
* **Parameters**:
  + **Number of Neurons**: Determines the network's capacity (e.g., 128, 64 neurons).
  + **Activation Function**: Typically ReLU for hidden layers and Softmax for output.
* **Output**: Probability distribution over gesture classes.

**Steps Involved in Using a CNN for Sign Language Detection:**

1. **Dataset Preparation**:  
   The CNN is trained using a dataset of labeled images representing various sign language gestures. These images may include diverse backgrounds, lighting conditions, and hand orientations to ensure robustness.
2. **Pre-processing**:  
   Input images are resized, normalized, and augmented (if needed) to improve the model's performance. This step ensures the data is consistent and that the CNN can generalize well across different scenarios.
3. **Feature Extraction (via Convolutional Layers)**:  
   The CNN extracts features from the input images through convolutional layers. These layers identify patterns such as edges, textures, and shapes in the hand's image.
4. **Pooling (Downsampling)**:  
   Pooling layers reduce the spatial dimensions of feature maps, retaining essential information while making computations efficient.
5. **Classification (via Fully Connected Layers)**:  
   The extracted features are passed to fully connected layers that assign probabilities to each gesture class. The output is a predicted gesture label or class ID corresponding to the detected sign.
6. **Output**:  
   The CNN outputs a label or ID that represents the recognized sign language gesture. This result can then be displayed to the user or integrated into a system for further interaction.

**Advantages of CNN for Sign Language Detection:**

* **High Accuracy**: CNNs can learn complex patterns, making them ideal for nuanced gestures.
* **Automatic Feature Learning**: Unlike traditional methods, CNNs do not require manual feature engineering.
* **Scalability**: CNNs can handle large datasets and complex tasks, making them suitable for recognizing a wide range of gestures.

# CHAPTER 4 IMPLEMENTATION AND TESTING

## 4.1 Implementation

### 4.1.1 Tools Used (Programming Languages)

The mostly used tools used in this applications is python,

* PYTHON

Python is a popular, high-level programming language that is widely utilized in diverse applications like scientific computing, web development, data analysis, artificial intelligence, and more. It supports a wide range of object-oriented programming (OOP) concepts, including encapsulation, inheritance, and polymorphism. Python also boasts a rich collection of third-party frameworks and standard libraries for various applications, making it a popular choice for developers and organizations.

* Microsoft Word

This tool is used to do all the documentation of our project from the scratch to the very end.

* Visual Studio Code

This is our code editor where we have written our all of codes. This tool is very user friendly and have lots of extensions which helps for making the coding process more efficient

* Microsoft PowerPoint

This tool is used to make the PowerPoint slides to do presentation of our project.

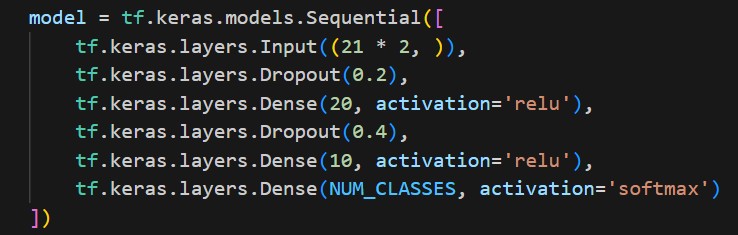
#### 4.1.1.1 Implementation Details of Modules (Description of Procedures/ functions)

**Gesture Recognition Module:** This module contains specialized system designed to identify and interpret gestures made with the hands using various technologies, primarily for human-computer interaction. This module typically integrates computer vision and machine learning techniques to detect and classify hand movements.

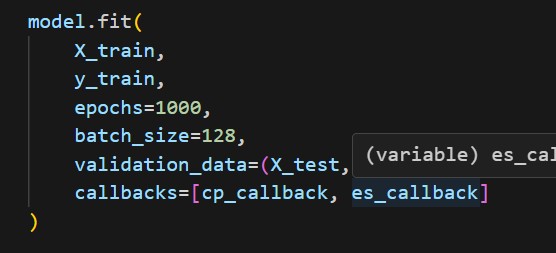


In these lines, the dataset is loaded from a CSV file (specified by the dataset variable) using NumPy’s loadtxt function. X\_dataset contains the feature data, which consists of 40 columns (21 \* 2, possibly representing 21 landmarks with x and y coordinates), while y\_dataset contains the labels, which represent the classes of gestures, stored in the first column of the dataset. The features are stored as float32 and the labels as int32.

The dataset is split into training and testing sets using train\_test\_split from Scikit-learn. The training set consists of 75% of the data (train\_size=0.75), and the testing set contains the remaining 25%. The random\_state parameter ensures that the split is reproducible.



In this part, a sequential neural network model is defined. The input layer expects an input shape of 40 (21 landmarks with x and y coordinates). Two dropout layers are included to prevent overfitting, where 20% and 40% of the neurons will be randomly ignored during training. The network contains two dense layers with 20 and 10 neurons respectively, both utilizing the ReLU activation function. The output layer uses softmax activation to predict the probability distribution across the defined number of classes (NUM\_CLASSES), which corresponds to the gesture classes.



The model is trained using the fit method. It trains on the X\_train and y\_train datasets for 1000 epochs with a batch size of 128. The validation\_data parameter allows the model to evaluate its performance on the test set after each epoch. The callbacks (cp\_callback and es\_callback) are used to save model checkpoints and implement early stopping, respectively, enhancing training efficiency and preventing overfitting.

## 4.2 Testing

Testing is the process of detecting the errors. It performs a very crucial role for quality assurance and for ensuring the reliability of the software. The results of testing are used later on during maintenance also. Testing requires a lot of time and labor.

### 4.2.1 Test Case for Unit Testing

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation.:

Table :Gesture Detection with valid Gestures

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S. N | Action | Steps | Expected  Outcomes | Actual  Outcomes | Test Result |
| 1 | Button press ‘Open camera’ | Press button open camera | “visual of camera” | No  visual | fail |
| 2 | Button press ‘Open camera’ | Press button open camera | “visual of camera” | “visual of camera” | pass |
| 3 | Detect "hello" gesture | Perform  "hello" gesture in front of the camera | "Hello" gesture recognized | No gesture | fail |
| 4 | Detect "hello" gesture | Perform  "Hello" gesture in front of the camera | "hello" gesture recognized | "hello" gesture recognized | Pass |
| 5 | Detect  "power" gesture | Perform  "power" gesture | "Power" gesture recognized | "Power" gesture recognized | Pass |
| 6 | Detect "rock" gesture | Perform "rock" gesture | "Rock" gesture recognized | "Rock" gesture recognized | Pass |
| 7 | Detect "good luck" gesture | Perform "good luck" gesture | "Good luck" gesture recognized | "Good luck" gesture recognized | Pass |
| 8 | Button press ‘quit’ | Press button quit camera | “close visual of camera” | “visual of camera” | fail |
| 9 | Button press ‘quit’ | Press button quit camera | “close visual of camera” | “close visual of camera” | pass |

### 4.2.2 Test Case for System Testing

System testing is an overall testing of the system after integrating all the functions of the project. When all the functions of the Sign language detection (python-based project) are integrated then system testing is done.

Table : Gesture Detection with Invalid Gestures

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S.N | Action | Input | Expected  Outcomes | Actual  Outcomes | Test Result |
| 1 | Button press ‘Open camera’ | Press button open camera | “visual of camera” | No  visual | fail |
| 2 | Button press ‘Open camera’ | Press button open camera | “visual of camera” | “visual of camera” | pass |
| 3 | Invalid gesture detection | Perform random or  untrained gesture in front of the camera | Gesture not recognized | Last Gesture recognized | fail |
| 4 | Invalid gesture detection | Perform random or  untrained gesture in front of the camera | Gesture not recognized | Gesture not recognized | Pass |
| 5 | Multiple gestures at once | Perform more than one gesture at the same time | No gesture recognized | Trys to capture both hand | fail |
| 6 | Multiple gestures at once | Perform more than one gesture at the same time | No gesture recognized | No gesture recognized | pass |

# CHAPTER 5 CONCLUSION AND FUTURE RECOMMENDATIONS

## 5.1 Lesson learnt/ Outcome

During the development of this application, the authors encountered several moments where they envisioned additional features that could enhance the system. However, due to limitations in technical expertise and the scope of the chosen programming languages, certain aspects could not be fully realized. These limitations highlight areas for future improvement, as the authors continue to explore and gain proficiency in this technology.

The project also presented a valuable learning curve, particularly in time management, as the authors had to balance development with documentation to meet strict deadlines. While the project successfully met its intended goals and expectations, the authors recognize potential opportunities for refinement. Moving forward, they aim to modify specific functions to improve usability and make the application more competitive and accessible for users.

## 5.2 Conclusion

In conclusion, the sign language detection project implemented in Python serves as a robust platform for users to interact with innovative applications designed to recognize and interpret sign language gestures. This project integrates multiple modules and functionalities to ensure smooth interaction and efficient management of gesture detection tasks.

By offering a dedicated platform for sign language detection, the project promotes accessibility and empowers users to engage more naturally with technology. The collaborative operation of its modules delivers an intuitive and seamless experience, showcasing the potential of gesture recognition in fostering improved communication and interaction in the digital space.

.

## 5.3 Future Recommendation

* As technology advances rapidly, our vision for the future of the sign language detection project is to provide enhanced user experience with outstanding service and accuracy. We aim to elevate the detection system by continuously refining its capabilities, responsiveness, and integration with various applications. This forward-thinking approach will empower users to interact effortlessly with technology, fostering a more intuitive and engaging interface for applications ranging from education to accessibility.
* In the future, the Python-based "Sign Language Detection System" can incorporate several enhancements to improve user experience further. Some potential advancements include:
* Expanding the repertoire of recognized gestures to increase usability across diverse scenarios.
* Enabling multi-hand detection and recognition for more complex sign language inputs.
* Increasing the diversity and volume of training data, including varying hand sizes, skin tones, and environmental conditions, to enhance model accuracy and robustness.
* Extending the system's functionality to mobile and web applications, ensuring greater accessibility for users on different platforms.
* These improvements will solidify the project's role as an innovative and practical solution for seamless communication and interaction in modern digital environments.

# REFERENCES

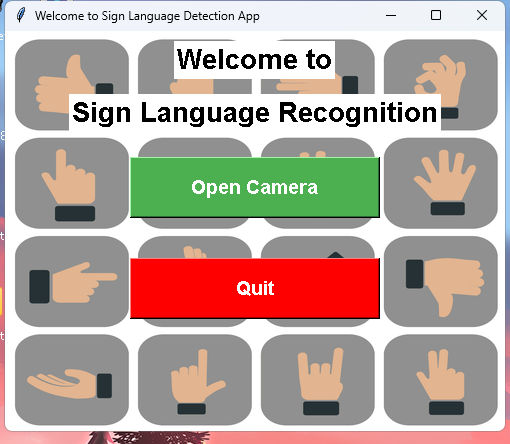
1. D. K. Ray, M. Soni, P. Johri, and A. Gupta, “Hand Gesture Recognition using Python,” *International Journal on Future Revolution in Computer Science & Communication Engineering*, vol. 4, no. 6, pp. 59–62, Jun. 2018, [Online]. Available: http://www.ijfrcsce.org/index.php/ijfrcsce/article/view/1687.
2. V. Harini, V. Prahelika, I. Sneka, and P. A. Ebenezer, “Hand gesture recognition using OpenCV and Python,” in Springer eBooks, 2020, pp. 1711–1719. doi: 10.1007/978-3-030-41862-5\_174.

[3 A. P. Ismail, F. A. A. Aziz, N. M. Kasim, and K. Daud, “Hand gesture recognition on python and opencv,” IOP Conference Series Materials Science and Engineering, vol. 1045, no. 1, p. 012043, Feb. 2021, doi: 10.1088/1757-899x/1045/1/012043.

[4] M. Y. Baihaqi, N. Vincent, and J. W. Simatupang, “Real-Time hand gesture recognition for humanoid robot control using Python CVZone,” in Lecture notes in networks and systems, 2023, pp. 262–271. doi: 10.1007/978-3-031-26852-6\_24.

# APPENDIX

##### Landing Page



Screenshot[1]:Landing page

##### Gestures

A blurry image of a room

Description automatically generated

Screenshot[2]:No sign



Screenshot[3]:Gesture(ak - one)



Screenshot[4]:Gesture(sabaii thik (all good)

A screenshot of a computer

Description automatically generated

Screenshot[5]:Gesture(Timi)

A screenshot of a computer

Description automatically generated

Screenshot[6]:Gesture(bandook - gun)



Screenshot[6]:Gesture(muthi)