HAND SIGN DETECTION USING COMPUTER VISION

A Mini Project Report Submitted

In partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

PULAVARTHI MANOJ KUMAR 21N31A05J9

POTHUGANTI YUGESHWAR 21N31A05J8

NALLA ABHINAV 21N31A05F7

Under the esteemed guidance of

Ms. V. Shilpa

Assistant Professor

Department of Computer Science and Engineering

Malla Reddy College of Engineering and Technology

(Autonomous Institution-UGC, Govt. of India)

(Affiliated to JNTUH, Hyderabad, Approved by AICTE, NBA&NAAC with 'A' Grade)

Maisammaguda, Kompally, Dhulapally, Secunderabad – 500100

Website: www.mrcet.ac.in

2024-2025

Malla Reddy College of Engineering and Technology

(Autonomous Institution-UGC, Govt. of India)

(Affiliated to JNTUH, Hyderabad, Approved by AICTE, NBA&NAAC with 'A' Grade)
Maisammaguda, Kompally, Dhulapally, Secunderabad – 500100
Website: www.mrcet.ac.in

CERTIFICATE

This is to certify that this is the Bonafide record of the Mini project entitled **"Hand Sign Detection Using Computer Vision"**, submitted by **PULAVARTHI MANOJ KUMAR (21N31A05J9), POTHUGANTI YUGESHWAR (21N31A05J8), NALLA ABHINAV (21N31A05F7)** of B. Tech in the partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science and Engineering Department of CSE during the year 2024-25. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Guide
Ms. V. Shilpa
Assistant Professor

Head of the Department
Dr. S. Shanthi
Professor

External Examiner

DECLARATION

We hereby declare that the project titled "**Hand Sign Detection Using Computer Vision"** submitted to Malla Reddy College of Engineering and Technology (UGC Autonomous), affiliated to Jawaharlal Nehru Technological University Hyderabad (JNTUH)for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a result of original research carried-out in this thesis. It is further declared that the project report or any part thereof has not been previously submitted to any University or Institute for the award of degree or diploma.

PULAVARTHI MANOJ KUMAR - 21N31A05J9

POTHUGANTI YUGESHWAR - 21N31A05J8

NALLA ABHINAV - 21N31A05F7

ACKNOWLEDGEMENT

We feel honored to place my warm salutation to our college Malla Reddy College of Engineering and Technology (UGC-Autonomous) for allowing us to do this Project as part of our B. Tech Program. We are ever grateful to our Director **Dr. V. S. K Reddy** and Principal **Dr. S. Srinivasa Rao** who enabled us to have experience in engineering and gain profound technical knowledge.

We express my heartiest thanks to our HOD, **Dr. S. Shanthi** for encouraging using every aspect of our course and helping us realize our full potential.

We would like to thank our Project Guide **Ms. V.Shilpa** and Project Coordinator **Ms. R. Sujatha** for her regular guidance, suggestions, constant encouragement, continuous monitoring, and unflinching cooperation throughout project work.

We would like to thank our Class Incharge **Ms. R. Sujatha** who despite being busy with her academic duties took time to guide and keep us on the correct path.

We would also like to thank all the faculty members and supporting staff of the Department of CSE and all other departments who have helped directly or indirectly in making our project a success.

We are extremely grateful to our parents for their blessings and prayers for the completion of our project which gave us the strength to do our project.

With regards and gratitude

PULAVARTHI MANOJ KUMAR - 21N31A05J9

POTHUGANTI YUGESHWAR - 21N31A05J8

NALLA ABHINAV - 21N31A05F7

ABSTRACT

Safety and Security are the major concerns in modern day era. Homes are frequently left unattended due to complex activities. Most individuals use closed circuit television to protect their homes when they are away from home. These systems are unaware of happenings in the surroundings as they just observe without any detection. It requires human interaction to work properly. We propose a Quick-Witted Security System using OpenCV. It includes features like Monitoring the theft, Detecting the noise. System also checks the visitors count and updates in and out count. This system comes with an extra feature of face recognition (Using OpenCV) where it identifies image of known and unknown faces by image processing. This system is developed to provide smart security in residential and work areas. Quick Witted Security System is a python GUI application which can run on any operating system, uses camera, and has number of features which are not in normal CCTV. This system comes with a feature of face recognition (Using OpenCV) where it identifies image of known and unknown faces by given data set.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF OUTPUTS

1. INTRODUCTION

The project "Hand Sign Detection Using Computer Vision in Python" aims to create an innovative and interactive system that can recognize and classify hand gestures in real-time using computer vision techniques. With the rise of natural user interfaces and the growing need for intuitive interaction methods, this project leverages Python's extensive libraries such as OpenCV and TensorFlow to build a system that processes hand movements from live video input and interprets them into meaningful actions or commands. The system's primary goal is to bridge the gap between humans and machines, allowing for touchless interaction in various applications.

This project's significance lies in its wide range of applications, from accessibility tools for the differently-abled to gesture-based controls in gaming, virtual reality, and smart home environments. The ability to recognize hand gestures in real-time also has practical implications for industries such as robotics, healthcare, and education, where touchless and gesture-based control can enhance user experience and functionality. By employing advanced image processing and deep learning models, the system can accurately detect hand signs in diverse conditions, making it adaptable and reliable for real-world scenarios.

The technical foundation of this project revolves around the integration of computer vision and machine learning techniques, enabling the system to learn and improve gesture recognition over time. The use of Python provides flexibility and access to a wide array of libraries that facilitate tasks such as video capture, image preprocessing, and neural network training. As a result, this project offers a scalable solution that can be implemented on different platforms and devices, opening the door to future enhancements and extensions, such as expanding gesture libraries or integrating with external systems like IoT or robotic platforms.

1.1 PURPOSE, AIM AND OBJECTIVES

The primary purpose of hand sign detection is to interpret human gestures and convert them into machine-readable commands. This capability allows users to interact with devices without the need for physical interfaces or controllers, fostering a more seamless and engaging user experience. The technology aims to enhance accessibility for individuals with disabilities and improve efficiency in environments where traditional input methods may be impractical.

1. **Natural Interaction:**

   Hand sign detection aims to facilitate a more natural way for humans to interact with machines, eliminating the need for physical controllers or interfaces.

2. **Accessibility**:

   It enhances accessibility for individuals with disabilities, allowing them to communicate and control devices using gestures instead of traditional input methods.

3. **Real-Time Communication:**

   The technology enables real-time gesture recognition, allowing for immediate feedback and interaction, which is essential in dynamic environments.

4. **Human-Computer Interaction (HCI):**

   It improves the quality of human-computer interactions by interpreting gestures as commands, making technology more intuitive and user-friendly.

5. **Versatile Applications:**

   Hand sign detection can be applied across various fields, including robotics, virtual reality, gaming, healthcare, and smart home automation.

6. **Gesture Recognition Accuracy:**

   The purpose includes achieving high accuracy in recognizing a wide range of gestures, which is critical for reliable user experiences.

7. **Machine Learning Integration:**

   The system utilizes machine learning algorithms to continuously improve gesture recognition capabilities through training on diverse datasets.

8. **Cost-Effective Solutions:**

   By replacing expensive hardware like data gloves with camera-based systems, hand sign detection provides cost-effective solutions for gesture recognition.

9. **Enhancing Communication:**

   It supports communication in sign languages, enabling better interaction between deaf individuals and technology or other users.

1.1 BACKGROUND OF PROJECT

Hand sign detection using computer vision has gained significant attention in recent years due to its potential to enhance human-computer interaction (HCI) and accessibility for individuals with disabilities. The concept of interpreting hand gestures to communicate has a longstanding history, particularly in the context of sign languages used by the deaf and hard-of-hearing communities. Traditional hand sign recognition relies on sensors, such as gloves equipped with sensors, which can accurately capture hand movements but are often costly, cumbersome, and impractical for everyday use. With advancements in computer vision and machine learning, it has become possible to create robust, non-intrusive systems that use standard cameras to detect and interpret hand gestures, making the technology accessible to a broader audience and opening up new applications.

Recent developments in deep learning and image processing have revolutionized the field of computer vision, enabling the creation of models capable of identifying objects, faces, and increasingly complex gestures with high precision. Convolutional Neural Networks (CNNs) and other deep learning architectures have been particularly effective in image recognition tasks, making them a natural choice for hand sign detection projects. By training these models on large, diverse datasets of hand signs, systems can learn to recognize a wide range of gestures even in challenging conditions. Additionally, advancements in real-time processing enable these models to be used in interactive applications, where quick response times are essential, allowing the system to perform efficiently on various devices.

Hand sign detection is not only a technical achievement but also offers social benefits, as it has the potential to bridge communication gaps and improve accessibility in numerous fields, including education, healthcare, and assistive technology. A reliable hand sign detection system could serve as a powerful tool for users with communication barriers, providing an intuitive

A reliable hand sign detection system could serve as a powerful tool for users with communication barriers, providing an intuitive way to interact with technology. Beyond accessibility, this technology can be applied to hands-free interfaces in everyday applications, such as smart home control, virtual reality, gaming, and safety monitoring systems.

1.2 SCOPE OF THE PROJECT

The scope of the project "Hand Sign Detection Using Computer Vision in Python" encompasses a wide range of potential applications and future enhancements. This project not only aims to provide a robust and efficient hand gesture recognition system but also highlights the various use cases in fields like human-computer interaction, accessibility, and entertainment. By leveraging Python's vast ecosystem of libraries, the project can be scaled and adapted to multiple platforms and devices. The scope further extends to potential improvements in accuracy, speed, and user experience, enabling the project to evolve alongside advancements in computer vision and machine learning. Here's a detailed breakdown of the project's scope:

1. **Human-Computer Interaction (HCI):** The project contributes to more natural and intuitive human-computer interactions by enabling gesture-based controls, allowing users to interact with devices without traditional input methods like keyboards or touchscreens.
2. **Assistive Technologies:** This project can improve accessibility for individuals with disabilities, especially in sign language interpretation, where the system can recognize and translate hand signs into readable text or speech in real-time.
3. **Gaming and Entertainment:** Hand sign detection systems can be applied to gesture-based gaming or virtual reality environments, allowing users to control in-game actions through hand movements, offering a more immersive gaming experience.
4. **Home Automation and IoT:** The system can be integrated into smart home environments, where users can control devices like lights, appliances, or media systems using hand gestures, enhancing the functionality of IoT systems.
5. **Robotics and Automation:** The project can be extended to control robotic systems through gestures, offering a more intuitive way to command robots in industrial automation, healthcare, or even personal assistant robots.
6. **Augmented Reality (AR) and Virtual Reality (VR):** The system can enhance AR and VR applications by allowing users to manipulate virtual objects, navigate virtual environments, or interact with immersive experiences using hand gestures, creating a seamless interaction.
7. **Sign Language Recognition:** One of the significant applications is in sign language recognition, where the system can be developed to recognize various sign languages, providing a bridge between hearing-impaired individuals and those unfamiliar with sign language.

8. **Education and Learning Tools:** Hand gesture detection can be incorporated into educational tools, allowing interactive learning experiences where students can use hand signs to engage with virtual learning environments or control visual aids during presentations.
9. **Mobile and Wearable Devices:** The project's scope extends to mobile and wearable devices where gesture recognition can be implemented for hands-free control of smartphones, tablets, or wearable devices like smartwatches and AR glasses.
10. **Scalability for Future Enhancements:** The project is scalable for future improvements, such as increasing the number of gestures the system can detect, enhancing accuracy under different environmental conditions, and incorporating multi-user detection in more complex scenario.

1.3 MODULE DESCRIPTION

1.4.1 DATACOLLECTION MODULE (DATACOLLECTION.PY)

- **Description**: This module is responsible for capturing images of hand signs and saving them for training purposes. It uses a webcam feed to detect a hand within the frame, crops and resizes the hand region to a standard size, and saves these processed images in a specified folder.
- **Responsibilities**:
    - Capture real-time video feed from the webcam.
    - Detect the presence of a hand using a hand detection model.
    - Crop and resize the detected hand region to a standard size for uniformity.
    - Save processed hand images to a designated directory.
- **Technologies Used**:
    - **OpenCV**: Used for video capture, image processing, and displaying the image feed.
    - **cvzone HandTrackingModule**: Used for hand detection and tracking in the video feed.
    - **NumPy**: Used for image array manipulations.
    - **Python's Math Module**: Utilized for size calculations when resizing images.
    - **Time Module**: Used to timestamp saved images for unique filenames.

1.4.1 TEST MODULE (TEST.PY)

- **Description**: This module tests the hand sign classification model. It captures a real-time video feed, detects hand signs, processes the hand region, and then uses a trained classification model to predict the specific hand sign. The prediction is displayed on the video feed in real time.
- **Responsibilities**:
    - Capture and display a live video feed from the webcam.
    - Detect a hand in the frame and crop the hand region.
    - Use a trained model to predict the hand sign from the cropped image.
    - Display the predicted label on the video feed in real time.
- **Technologies Used**:
    - **OpenCV**: For video capture and display of processed image with prediction results.
    - **cvzone HandTrackingModule**: Detects hand presence and location.
    - **cvzone ClassificationModule**: Loads and runs predictions using a pre-trained model.
    - **NumPy**: Manages array manipulations for resizing and padding images.
    - **Python's Math Module**: Used for calculations related to resizing.
    - **Time Module**: Utilized for controlling frame capture timing.

2. SYSTEM ANALYSIS

The system analysis of the "Hand Sign Detection Using Computer Vision in Python" project focuses on the architecture and workflow of the system. It processes real-time video input from a webcam, applying image preprocessing techniques like background subtraction and contour detection to isolate hand gestures.

2.1 HARDWARE AND SOFTWARE REQUIREMENTS

2.1.1 HARDWARE REQUIREMENTS

- Minimum 4Gb RAM
- Minimum 50 GB Hard Disk
- Minimum i5 processor

2.1.2 SOFTWARE REQUIREMENTS

- OS-Windows 10 or Mac
- OpenCV Technology
- Python Programming Language

2.2 SOFTWARE REQUIREMENT SPECIFICATION

2.2.1 SRS

The software requirements specification (SRS) for "Hand Sign Detection using Computer Vision in Python" includes the use of Python libraries like OpenCV for image processing, TensorFlow/Keras for deep learning, and Numpy for data handling. The system must run on platforms like Windows, macOS, or Linux, supporting real-time video processing and providing gesture-based interface control.

2.2.2 ROLE OF SRS:

The purpose of the Software Requirement Specification is to reduce the communication gap between the clients and the developers. Software Requirement Specification is the medium through which the client and user needs are accurately specified.

FUNCTIONAL REQUIREMENTS

1. **Real-Time Video Capture:** The system must capture live video feed from a webcam or external camera to detect hand gestures in real-time.
2. **Hand Detection and Tracking:** The system should accurately detect and track hand regions within the video frame using techniques like skin color detection, background subtraction, or contour detection.
3. **Preprocessing of Hand Images:** The system must preprocess the captured images (e.g., resizing, grayscale conversion, noise removal) to prepare them for gesture recognition.
4. **Gesture Recognition Model:** A trained deep learning model (e.g., CNN) must classify the detected hand gestures into predefined categories (e.g., thumbs up, fist, open palm) based on the input data.
5. **Feature Extraction:** The system must extract key features from the hand image, such as shape, contour, or specific points like fingertips, to assist in accurate gesture classification.
6. **Gesture Classification and Mapping:** The system should classify the recognized gesture and map it to a corresponding command or action, such as controlling a device or triggering an event.
7. **Feedback Display:** The system must display real-time feedback on the user interface, showing the detected gesture and the corresponding action taken.
8. **Customization of Gestures:** Users should be able to add new gestures or customize existing gestures in the system by training the model on additional datasets.
9. **Error Detection and Recovery:** The system should detect and handle errors, such as false positives or unrecognized gestures, providing the user with options for recovery or retraining the model.
10. **Integration with External Systems:** The system must support integration with external applications, such as IoT devices, AR/VR platforms, or robotics, enabling gesture-based control for various systems.

INSTALLATION

PYTHON SETUP:

1. **INSTALL PYTHON PACKAGES:**
    1. Tensorflow
    2. Keras
    3. Numpy
    4. Hand Detector
    5. Math
    6. Time

2. TRAINING THE NETWORK

1. Data Collection: Use DataCollection.py
2. Testing: Use Test.py
3. Commands to install packages if not pre-installed:

    pip install opencv (or) pip install cvzone

    pip install tensorflow

    pip install keras

    pip install numpy

    pip install hand detector

    pip install math

    pip install time

NON-FUNCTIONAL REQUIREMENTS

1. **Performance:** The system should process video frames and classify hand gestures in real-time with minimal latency to ensure smooth interaction.
2. **Scalability:** The solution must be scalable, allowing it to handle an increasing number of gestures, users, or integration with more complex systems without significant performance degradation.
3. **Usability:** The interface must be intuitive and easy for users to understand and navigate, even without technical expertise.
4. **Accuracy:** The system should maintain high accuracy in gesture recognition, even in varying lighting conditions, different skin tones, and complex backgrounds.
5. **Security:** If the system captures personal data (e.g., live video), it should ensure the protection and privacy of the user's information.

6. **Maintainability:** The system must be designed with modularity in mind, making it easy to update, modify, or fix components such as gesture libraries or algorithms.
7. **Portability:** The system should be portable and work across different platforms (Windows, macOS, Linux), with minimal changes to the underlying code.
8. **Reliability:** The system should be reliable, performing consistently under different conditions, such as varying hand sizes, positions, and movement speeds.
9. **Resource Efficiency:** The system should efficiently use system resources (CPU, memory) to avoid excessive load, enabling smooth real-time processing even on low-end devices.
10. **Response Time:** The system should provide immediate feedback upon detecting a gesture, with a response time of less than 100 milliseconds to ensure seamless user interaction.

2.2.3 SCOPE:

This scope includes developing a machine learning model to accurately detect various hand gestures under different conditions, such as varying lighting and hand orientations. It also involves preprocessing techniques, real-time video handling, and efficient model inference to enable smooth user interaction.

The system can be applied in areas like sign language interpretation, gesture-based controls, and interactive applications, enhancing accessibility and human-computer interaction. The project will focus on accuracy, responsiveness, and user adaptability.

2.2.4 EXISTING SYSTEM:

The current or existing systems in hand sign detection may utilize traditional image processing techniques or older machine learning models, like:

Thresholding and Contour Detection: Simple methods to identify hand regions based on color, background subtraction, or edge detection.

Template Matching:

Using predefined templates or shapes to match hand gestures.

Basic Machine Learning Algorithms:

Some systems use classical algorithms like K-Nearest Neighbors (KNN) or Support Vector Machines (SVM) with hand-crafted features.

Non-Deep Learning Approaches:

Some older systems might avoid deep learning, using basic image processing and feature extraction techniques, like Histogram of Oriented Gradients (HOG) or Haar Cascades, to recognize gestures.

2.2.4.1 DRAWBACKS OF EXISTING SYSTEM:

The existing systems often suffer from several limitations, especially when compared to more advanced, deep learning-based methods. The key drawbacks include:

Limited Accuracy: Traditional methods are often less accurate and struggle with distinguishing complex or subtle gestures.

Sensitivity to Lighting and Background Variations: They are prone to errors under different lighting conditions, backgrounds, or changes in hand orientation.

High False Positives/Negatives: Existing systems may often misclassify non-gesture movements as gestures or fail to recognize gestures accurately.

Difficulty with Real-Time Processing: Many traditional algorithms are not optimized for real-Time performance, leading to latency or lag, which is critical for applications requiring Immediate feedback.

Limited Robustness to Variation: Basic systems are less flexible in handling variations in hand sizes, skin colors, or positions, reducing their effectiveness across diverse users.

Manual Feature Engineering: Traditional methods rely on manual feature extraction (e.g., edges, contours), which can be tedious and less effective than automated feature extraction using deep learning.

Inability to Handle Complex Gestures: They struggle with multi-finger or complex hand Gestures, which require advanced spatial understanding and often need deeper contextual analysis.

2.2.5 PROPOSED SYSTEM

Proposed System for Hand Sign Detection Using Computer Vision:

The proposed system for hand sign detection leverages advanced computer vision and machine learning techniques to identify and interpret various hand signs. This system typically consists of several components:

Image Acquisition: Capturing real-time video input from a camera or webcam.

Preprocessing: Enhancing image quality by applying techniques such as filtering, normalization, and resizing to prepare for model input.

Feature Extraction: Using methods like contour detection or deep learning-based feature extraction (e.g., CNNs) to identify relevant features from hand images.

Hand Sign Classification: Implementing machine learning algorithms (e.g., neural networks) to classify the extracted features into predefined hand signs.

2.2.5.1 ADVANTAGES OF PROPOSED SYSTEM:

Real-Time Detection: Enables real-time hand sign detection and interpretation, making it responsive and suitable for interactive applications.

Hands-Free Interaction: Provides a non-contact method of interaction, ideal for situations where physical interaction with a device is limited or inconvenient.

Accessibility Enhancement: Can serve as an assistive tool for people with speech or physical disabilities, facilitating communication through hand signs.

3. TECHNOLOGIES USED

3.1 MACHINE LEARNING

Machine learning, a subset of artificial intelligence, empowers computers to learn from data and make decisions or predictions without explicit programming. It revolves around developing algorithms and statistical models that can identify patterns and relationships within data. Unlike traditional programming, which relies on predetermined rules, machine learning algorithms refine their performance as they encounter more data, continually learning from experience. At the core of machine learning lies the process of training models on data. This process involves several essential steps, beginning with data collection. Data, sourced from various repositories such as sensors, databases, or text documents, serves as the fundamental building block for training and evaluating machine learning models. Once collected, the data undergoes preprocessing to cleanse it of noise, handle missing values, and normalize features, ensuring it is in an optimal format for training. Feature engineering is another crucial step where the input variables, or features, are selected or transformed to enhance the model's ability to capture relevant information from the data. This step plays a pivotal role in improving the model's generalization performance. Subsequently, the appropriate machine learning algorithm is selected based on the problem's characteristics and requirements.

There are three subcategories of machine learning:

1. Supervised machine learning models are trained with labeled data sets, which allow the models to learn and grow more accurate over time. For example, an algorithm would be trained with pictures of dogs and other things, all labeled by humans, and the machine would learn ways to identify pictures of dogs on its own. Supervised machine learning is the most common type used today.
2. In unsupervised machine learning, a program looks for patterns in unlabeled data. Unsupervised machine learning can find patterns or trends that people aren’t explicitly looking for. For example, an unsupervised machine learning program could look through online sales data and identify different types of clients making purchases.

3. Reinforcement machine learning trains machines through trial and error to take the best action by establishing a reward system. Reinforcement learning can train models to play games or train autonomous vehicles to drive by telling the machine when it made the right decisions, which helps it learn over time what actions it should take.

3.2 PYTHON

Python, an open-source programming language, has emerged as a versatile and powerful tool since its inception by Dutch programmer Guido van Rossum in 1991. Named after the British comedy series "Monty Python's Flying Circus," Python was designed to be both easy to read and efficient, combining simplicity with robustness. Over the years, Python has evolved into one of the most popular languages globally, owing to its readability, flexibility, and extensive libraries.

Well, these are just the minor points from our sides Python is just a lot more than this. Some more notes about python, Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently. There are two major Python versions: Python 2 and Python 3

3.3 COMPUTER VISION

Computer vision is a field of artificial intelligence (AI) that focuses on enabling machines to interpret and understand visual data from the world, such as images and videos. It aims to replicate the capabilities of human vision in computers, allowing systems to analyse and make decisions based on visual inputs. Through a combination of image processing, pattern recognition, and machine learning, computer vision allows machines to detect objects, track movements, and classify images. Its goal is to automate tasks that require visual understanding, making it a critical tool in various technological applications.

Computer vision relies on several key techniques to extract meaningful information from visual data. Image processing involves enhancing images or filtering out noise to make important features more visible.

Object detection and segmentation help identify and locate objects within an image, while feature extraction pulls out distinctive patterns like edges, shapes, or textures. Recently, deep learning models, particularly Convolutional Neural Networks (CNNs)**,** have revolutionized the field by allowing computers to automatically learn these features from large datasets, leading to more accurate and scalable systems for tasks like face recognition, image classification, and object tracking.

3.4 PACKAGES AND LIBRARIES

OpenCV

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python can process the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

CVZone

CVZone is an open-source Python library designed to simplify the process of developing computer vision applications. Built on top of OpenCV and Mediapipe, CVZone provides an easy-to-use interface for handling complex tasks like object detection, hand and face tracking, and pose estimation. It abstracts many of the lower-level details, making it more accessible for developers, especially those new to computer vision. CVZone also integrates key machine learning features, allowing users to develop interactive systems quickly with minimal code. The library is particularly popular for real-time applications and interactive projects, such as gesture-based control and augmented reality.

Time

The time module in Python provides functions to work with time-related tasks, such as retrieving the current time, pausing execution, or measuring the duration of code execution. It enables interaction with system-level time features, allowing precise control over program execution delays or handling timestamps.

Math

The math module in Python provides a comprehensive collection of mathematical functions that allow users to perform complex mathematical operations with ease. It includes functions for basic arithmetic, algebraic, trigonometric, logarithmic, and exponential operations. The module is essential for tasks that require high-precision calculations and offers support for handling floating-point numbers, constants like pi and e, and more. By leveraging these functions, Python programmers can execute efficient and reliable mathematical computations without the need for external libraries.

NumPy

NumPy (Numerical Python) is a powerful Python library used for scientific computing, providing support for large, multi-dimensional arrays and matrices, along with a wide range of mathematical functions to operate on these arrays. It is a foundational package for data manipulation and analysis, particularly useful in fields like data science, machine learning, and scientific research. NumPy enables efficient computation and manipulation of arrays, facilitating faster processing for numerical operations compared to standard Python lists, and is integral to many other libraries like Pandas, SciPy, and TensorFlow.

4. SYSTEM DESIGN & UML DIAGRAMS

4.1 SOFTWARE DESIGN

Designing software for a hand sign detection project using computer vision involves multiple layers, from capturing the input images to classifying the detected hand signs accurately. The first component in the design is the image acquisition and preprocessing module. Here, the system captures images or frames from a live video feed using a camera, ensuring high-quality and consistent image inputs. Preprocessing techniques such as resizing, grayscale conversion, and noise reduction are applied to standardize the images, making them easier for the system to process. Additionally, hand segmentation and background subtraction techniques are utilized to isolate the hand from the background. This helps to reduce computational complexity and improves detection accuracy. OpenCV is a popular library for these steps, as it provides robust methods for image processing tasks.

The next major component is the feature extraction and classification module. This step involves using a machine learning or deep learning model to recognize specific hand signs based on key features of the hand's position, shape, and gesture. Techniques such as Convolutional Neural Networks (CNNs) are well-suited for this, as they can automatically learn spatial hierarchies and patterns within the images. A custom dataset of hand signs can be created, or pre-existing datasets can be used to train the model. Once trained, the classifier is integrated with the preprocessing module, enabling real-time prediction. The final design may include a user interface for displaying the recognized hand sign and additional options for retraining the model if new signs need to be added. This modular approach ensures flexibility, accuracy, and scalability for future enhancements in the hand sign detection project.

4.2 ARCHITECTURE

This concept is like discussions regarding machine learning and its algorithms. For generalization and to examine its specifics, neural networks are primarily used.

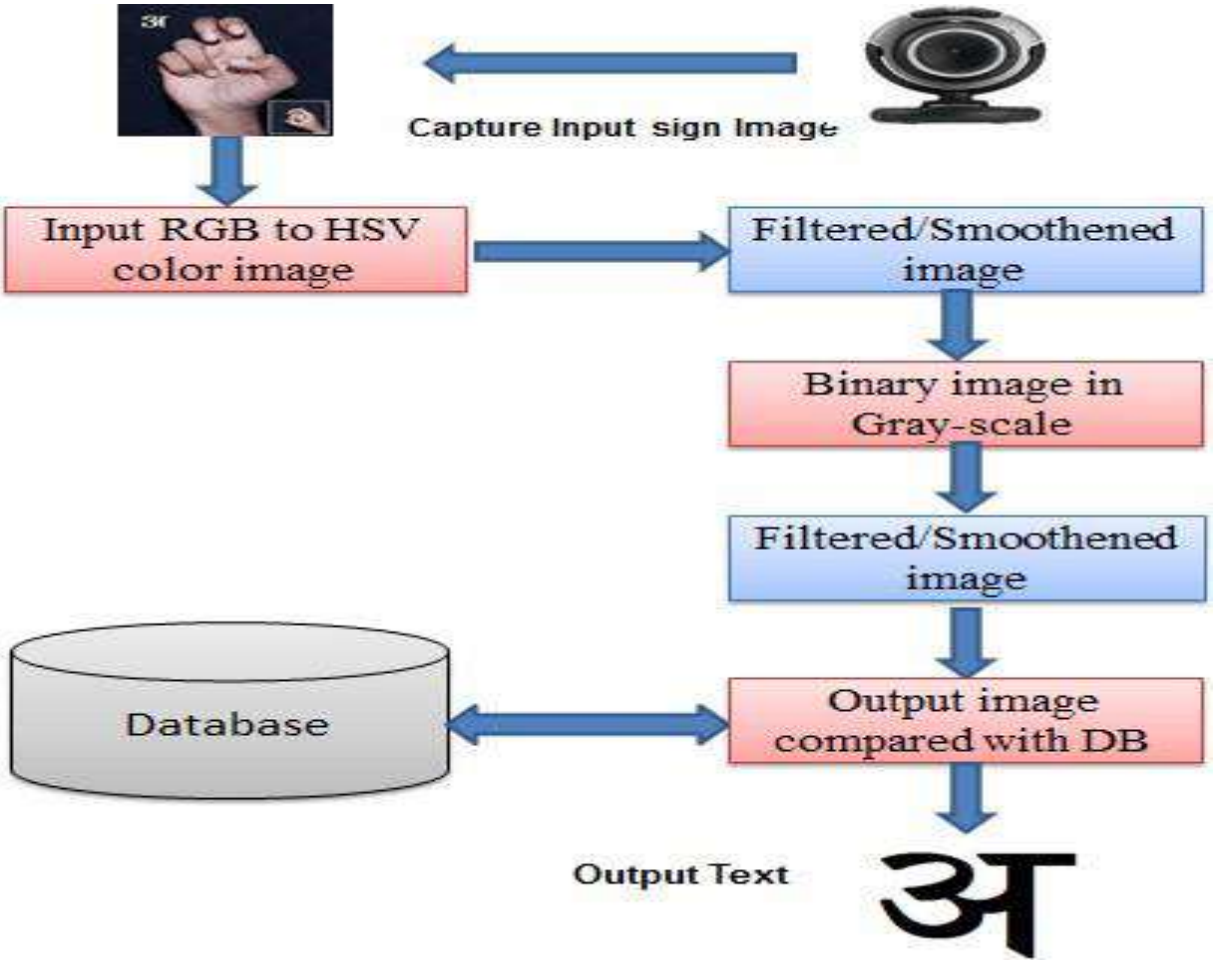

Fig 4.2.– System Architecture

4.3 UML DIAGRAMS

Unified Modeling Language (UML) diagrams are graphical representations used in software engineering to visualize the structure, behavior, and interactions of a system. These diagrams help in understanding, designing, and communicating the system's architecture and functionality among stakeholders.

UML diagrams can be categorized into several types, including structural diagrams such as class diagrams, object diagrams, and component diagrams, which depict the static structure of the system, including classes, objects, and their relationships. On the other hand, behavioral diagrams such as sequence diagrams, activity diagrams, and state diagrams illustrate the dynamic behavior and interactions within the system, showcasing how components collaborate to achieve specific functionalities.

A UML diagram is a way to visualize systems and software using Unified Modeling Language (UML). Software engineers create UML diagrams to understand the designs, code architecture, and proposed implementation of complex software systems. UML diagrams are also used to model workflows and business processes.

Additionally, deployment diagrams depict the physical deployment of software components across hardware nodes, while use case diagrams provide a high-level view of the system's functionalities from the user's perspective. Each UML diagram serves a specific purpose in capturing different aspects of the system, and collectively they offer a comprehensive representation of the system's architecture, behavior, and interactions.

4.3.1 Data Flow Diagram

A data-flow diagram is a visual representation of how data moves through a system or a process (usually an information system). The data flow diagram also shows the inputs and outputs of each entity as well as the process itself. A data-flow diagram lacks control flow, loops, and decision- making processes. With a flowchart, certain operations based on the data can be depicted. The flowchart can be used to understand how the data flows in this project. A video clip from a PTZ camera is used as the input in this case, and the number of frames on which the algorithm operates will later be converted. The result will be a picture in which a drone is recognized and shown using a rectangle frame around it.

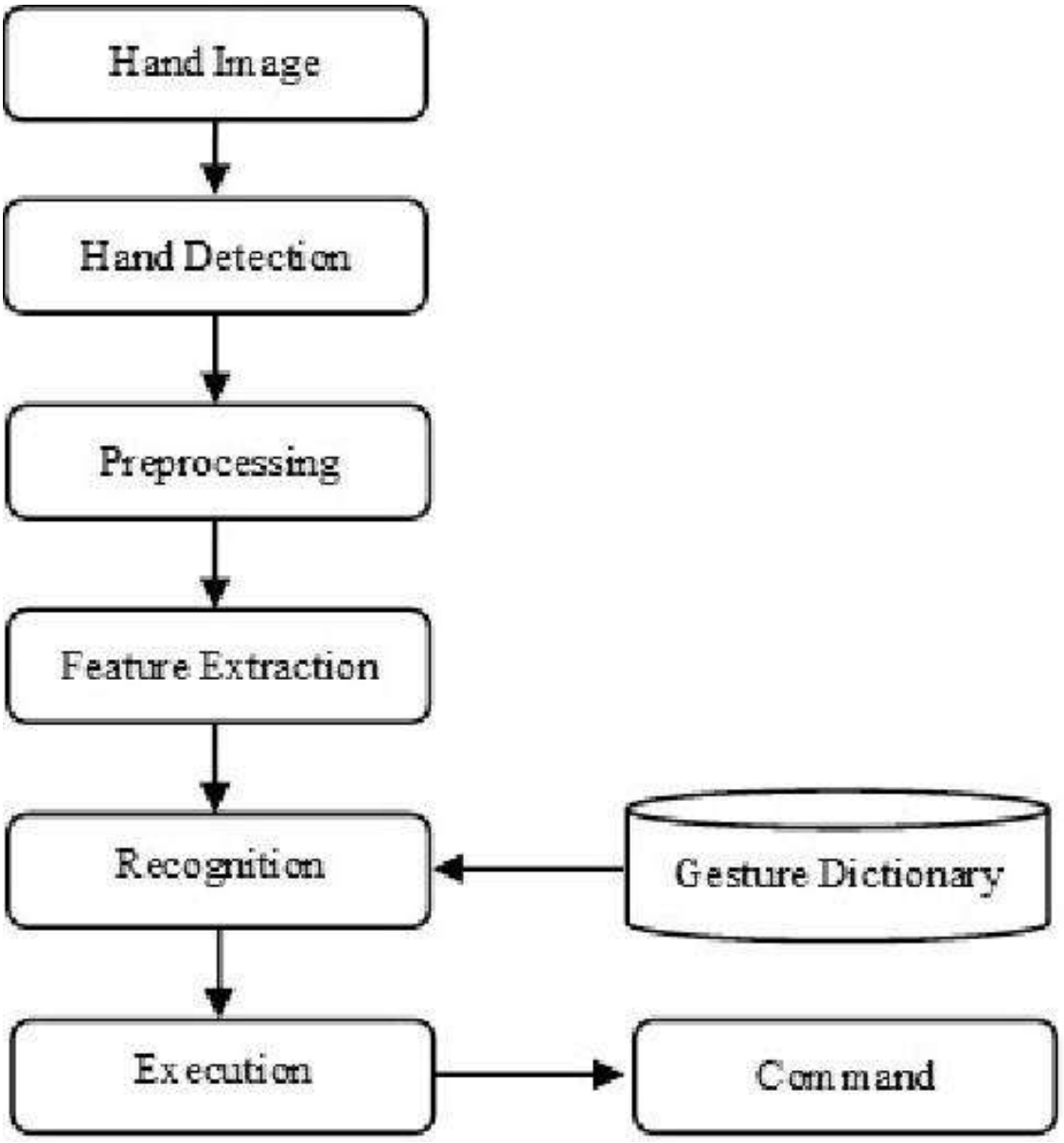

Fig. 4.3.1 Data Flow Diagram

4.3.2 Use Case Diagram

To depict a system's dynamic behavior, use case diagrams are often employed. Using use cases, actors, and their interactions, it captures the functionality of the system. A system or subsystem of an application's necessary duties, services, and operations are modelled. It shows a system's high- level functionality as well as how a user interacts with that system.

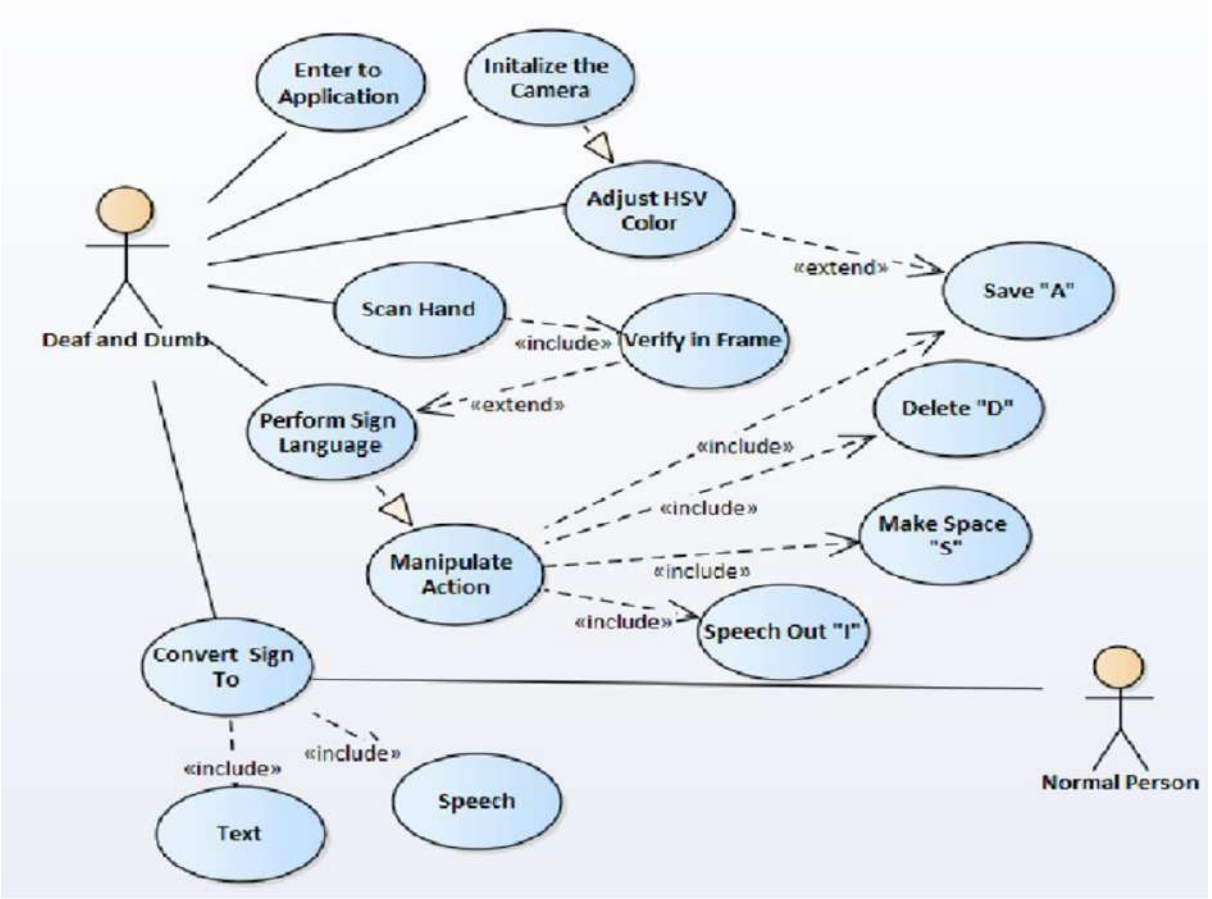

Fig. 4.3.2 Use Case Diagram

4.3.3 Class Diagram

Class diagrams are the main building block of any object-oriented solution. It displays a system's classes, along with each class's properties, operations, and relationships to other classes. Most modelling tools include three elements to a class. Name is at the top, followed by attributes, then operations or methods, and finally, methods. Classes are linked together to generate class diagrams in a complex system with numerous related classes. Various sorts of arrows represent different relationships between classes.

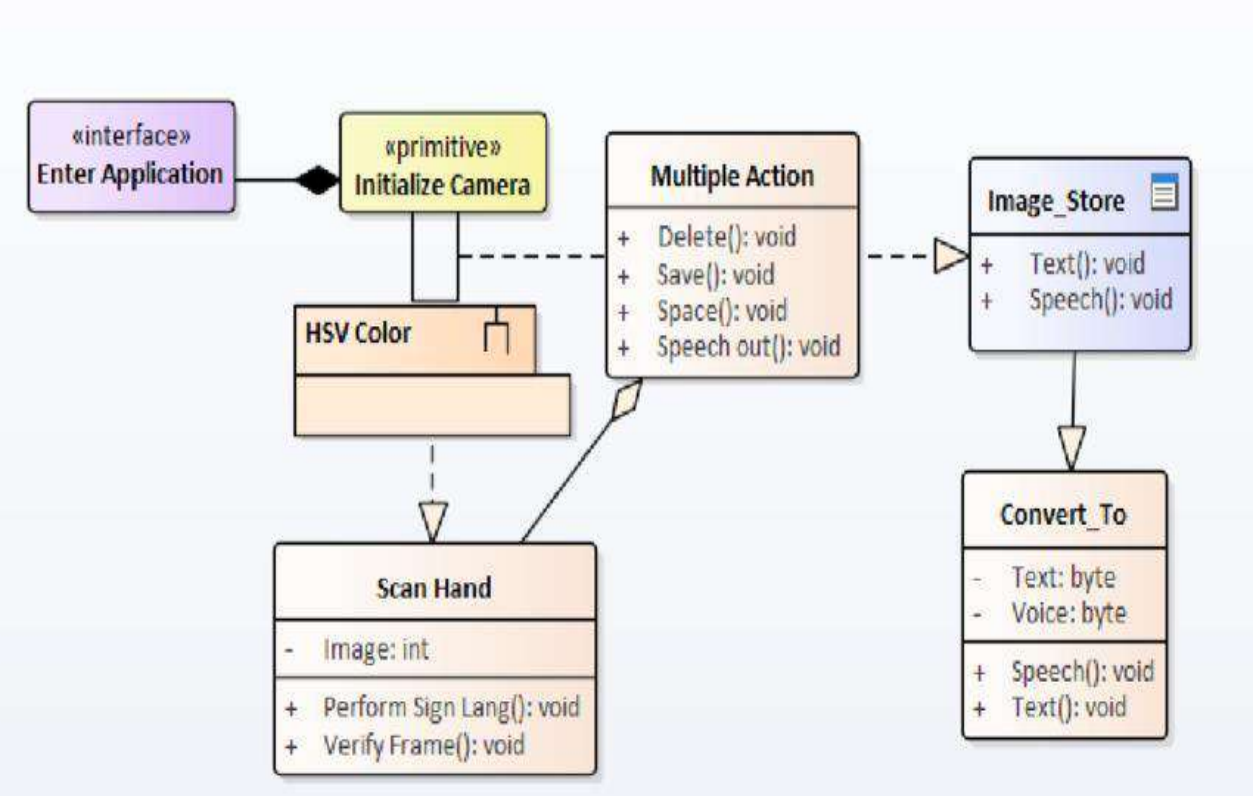

Fig. 4.3.3 Class Diagram

4.3.4 Sequence Diagram

In UML, sequence diagrams display how and in what order certain items interact with one another. It's crucial to remember that they depict the interactions for a certain circumstance. The interactions are depicted as arrows, while the processes are portrayed vertically. The objective of sequence diagrams and their fundamentals are explained in this article. To understand more about sequence diagrams, you may also look at this comprehensive tutorial.

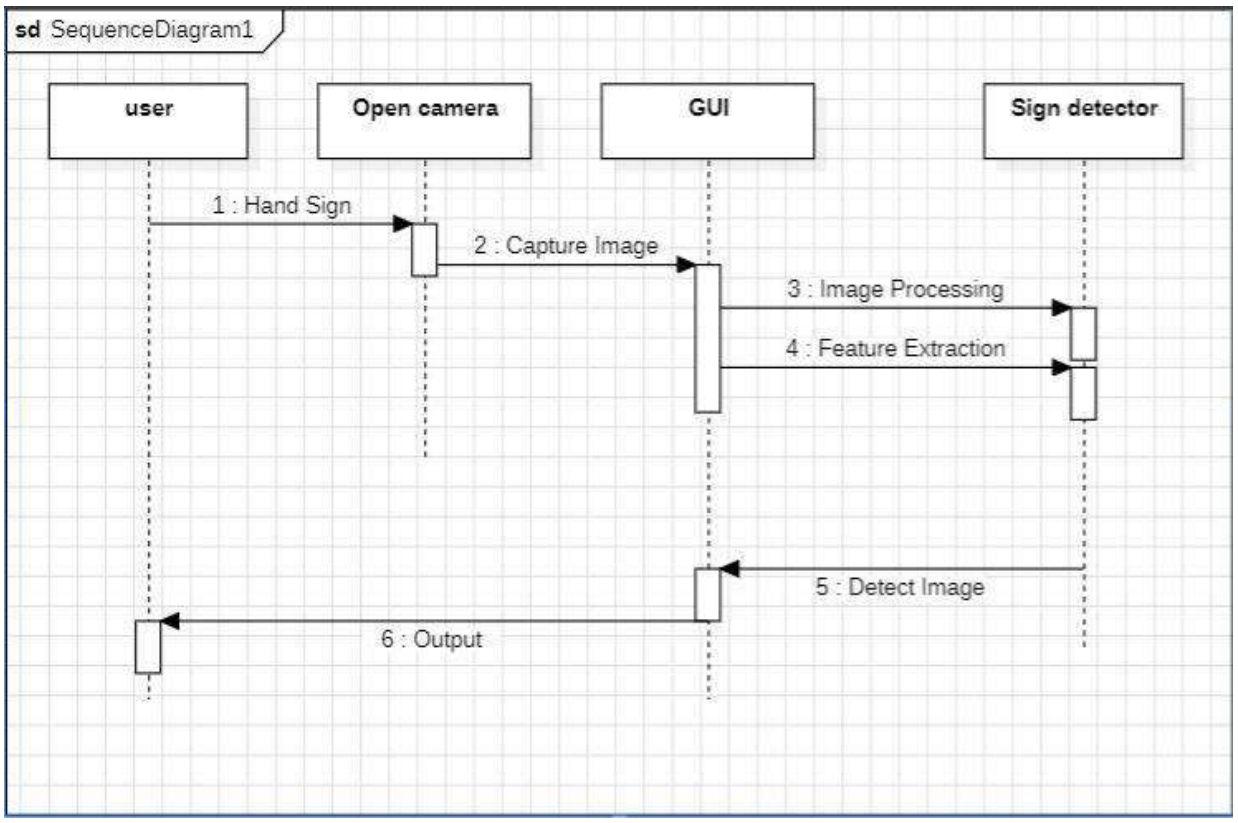

Fig. 4.3.4 Sequence Diagram

5. INPUT/OUTPUT DESIGN

5.1 INPUT DESIGN

The input design in any system focuses on how data is captured and prepared for processing. In the context of the "Hand Sign Detection Using Computer Vision in Python" project, the input design involves capturing real-time video streams from a camera or webcam. The system must ensure that the input (hand gestures) is received in a format that can be efficiently processed. To achieve this, various preprocessing techniques are applied, such as image filtering, background subtraction, and colour thresholding, to isolate the hand from the background and remove noise. This step is crucial for improving the accuracy and speed of gesture detection.

The design of the input process must account for real-world conditions such as varying lighting, different hand shapes and sizes, and complex backgrounds. The system needs to be robust enough to handle these variations without compromising performance. Additionally, the input design ensures that the captured data is structured correctly before it is passed to the gesture recognition model. This includes resizing images, converting colour formats, and normalizing pixel values. The goal is to prepare clean, high-quality inputs that can enhance the system's ability to accurately detect and classify hand gestures in real time.

5.2 OUTPUT DESIGN

In the "Hand Sign Detection Using Computer Vision in Python" project, the output design focuses on how the system presents the results of gesture recognition to the user. The recognized hand gesture is displayed in real-time, often through a graphical user interface (GUI). The output may include visual feedback, such as displaying the name or symbol of the recognized gesture on the screen, or triggering specific actions based on the detected gesture, like controlling a device or interacting with an application. The design ensures that the output is clear, responsive, and user-friendly, providing immediate feedback to the user.

Beyond displaying recognized gestures, the output design may also involve integration with external systems or devices. For example, the output could control a smart device (e.g., turning on lights) or send commands to a robot based on the recognized hand sign.

6 IMPLEMENTATION

CODE:

DataCollection.py

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
import math
import time
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
offset = 20
imgSize = 300
folder = "D:/Programming/2) Mango Python/6) Experimental/Project - 4/Numbers/5"
counter = 0
while True:
    success, img = cap.read() #
    hands, img = detector.findHands(img) #
    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 225
        imgCrop = img[y - offset: y + h + offset, x - offset: x + w + offset]
        imgCropShape = imgCrop.shape
        aspectRatio = h / w
        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            imgResizeShape = imgResize.shape
            imgWhite[0:imgResizeShape[0], 0:imgResizeShape[1]] = imgResize
            wGap = math.ceil((imgSize-wCal)/2)
            imgWhite[:, wGap:wCal+wGap] = imgResize
```

```
    else:
        k = imgSize / w
        hCal = math.ceil(k * h)
        imgResize = cv2.resize(imgCrop, (imgSize, hCal))
        imgResizeShape = imgResize.shape
        hGap = math.ceil((imgSize - hCal) / 2)
        imgWhite[hGap:hCal + hGap, :] = imgResize
    # cv2.imshow("ImageCrop", imgCrop)
    # cv2.imshow("ImageWhite", imgWhite)
cv2.imshow("Image", img)
key = cv2.waitKey(1)
if key == ord("s") :
    counter += 1
    cv2.imwrite(f'{folder}/Image_{time.time()}.jpg',imgWhite)
    print(counter)
```

Test.py

```
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import math
import time
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
classifier = Classifier("D:/Programming/2)Python/6) Experimental/Project - 4/Model - Numbers/converted_keras/keras_model.h5","D:/Programming/2)Python/6) Experimental/Project - 4/Model - Numbers/converted_keras/labels.txt")
#classifier1 = Classifier("D:/Programming/2) Mango Python/6) Experimental/Project - 4/Model - E, F, G, H/converted_keras/keras_model.h5","D:/Programming/2) Mango Python/6) Experimental/Project - 4/Model - E, F, G, H/converted_keras/labels.txt")
offset = 20
imgSize = 300
folder = "D:/Programming/2)Python/6) Experimental/Project - 4/Model - Numbers"
counter = 0
# labels = ['Danger','Slow','Stop','Straight','U-Turn']
# labels = ['A","B","C","D","E","F","G","H","I","J","K","L","O","P","R","S","T","U","V","W","Y"]
labels = ["0","1","2","3","4","5"]
# labels = ['stop']
while True:
    success, img = cap.read()
    imgOutput = img.copy()
    hands, img = detector.findHands(img)
    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 225
        imgCrop = img[y - offset: y + h + offset, x - offset: x + w + offset]
```

```
        imgCropShape = imgCrop.shape
        aspectRatio = h / w
        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            imgResizeShape = imgResize.shape
            imgWhite[0:imgResizeShape[0], 0:imgResizeShape[1]] = imgResize
            wGap = math.ceil((imgSize-wCal)/2)
            imgWhite[:, wGap:wCal+wGap] = imgResize
            prediction, index = classifier.getPrediction(imgWhite,draw = False)
            #prediction, index = classifier1.getPrediction(imgWhite, draw=False)
            print(prediction, index)
        else:
            k = imgSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))
            imgResizeShape = imgResize.shape
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hCal + hGap, :] = imgResize
            prediction, index = classifier.getPrediction(imgWhite,draw = False)
            #prediction, index = classifier1.getPrediction(imgWhite, draw=False)

        # cv2.rectangle(imgOutput, (x - offset, y - offset - 50), (x - offset + 90, y - offset-50+50),
(255, 0 , 255), cv2.FILLED)
        cv2.putText(imgOutput, labels[index], (x, y - 26), cv2.FONT_HERSHEY_COMPLEX, 1.7,
(255, 255, 255), 2)
        cv2.rectangle(imgOutput, (x-offset, y-offset), (x +w + offset, y + h + offset), (255, 0, 255), 4)
        # cv2.imshow("ImageCrop", imgCrop)
        # cv2.imshow("ImageWhite", imgWhite)
    cv2.imshow("Image", imgOutput)
    cv2.waitKey(1)
```

7. TESTING

7.1 TESTING OBJECTIVES

In a hand sign detection project, the testing objectives focus on ensuring accuracy, reliability, and efficiency in detecting and recognizing hand signs. Key objectives include:

Accuracy Testing: Verify that the model accurately identifies various hand signs with high precision and recall.

Robustness Testing: Test the model's resilience to variations in hand sizes, angles, lighting, and backgrounds.

Real-time Performance: Ensure the detection is fast enough to work in real-time, with minimal latency.

False Positives/Negatives Reduction: Minimize misclassifications to avoid incorrect interpretations of hand signs.

Scalability: Check the model's ability to detect hand signs under different hardware environments (e.g., mobile, desktop).

7.2 TESTING METHODOLOGIES

To ensure comprehensive coverage, several testing methodologies can be applied:

Unit Testing: Test individual components, like preprocessing functions, feature extractors, and individual neural network layers, to ensure each part works correctly.

Integration Testing: Test the system as a whole, integrating each module (like camera input, preprocessing, and model inference) to confirm they work seamlessly together.

Performance Testing: Evaluate how the model performs in different environments, measuring latency, response time, and memory usage, especially under real-time conditions.

Stress Testing: Test the model's performance under demanding conditions, such as when there are multiple hands in the frame or when a user rapidly changes hand signs.

User Acceptance Testing (UAT): Engage with users who might use the application (e.g., people familiar with hand signs) to get feedback on the usability, accuracy, and practicality of the detection.

7.3 USER TRAINING

User training is crucial for maximizing the software's effectiveness and accuracy. It might include:

Guiding Users on Optimal Hand Positioning: Train users to present their hand at specific distances, angles, and lighting conditions for accurate detection.

Providing a User Manual: Create a manual or tutorial that explains how to use the application, interpret its output, and adjust settings if needed.

Demonstration of Common Use Cases: Show examples of each hand sign and how the software recognizes them, possibly including error cases and corrective actions.

Feedback Mechanism: Include a way for users to report issues, such as misclassified signs, which can help improve the model.

7.4 MAINTENANCE

The model requires regular maintenance to ensure sustained accuracy and usability:

Model Retraining: Periodically retrain the model on new data to adapt to variations or introduce additional hand signs.

Updating Dependencies: As libraries (e.g., OpenCV, TensorFlow) receive updates, compatibility must be checked to avoid issues.

Bug Fixes: Address any emerging bugs from user feedback or new hardware/software updates.

Performance Optimization: Continually improve real-time performance, especially if deploying on devices with limited resources.

Adding New Features: Maintenance might also include expanding hand sign vocabulary or adding multi-hand detection capabilities.

7.5 TESTING STRATEGY

The testing strategy should outline a structured approach to testing all aspects of the project:

Data Validation: Confirm the training and testing datasets are clean, balanced, and representative of the signs to be detected.

Model Validation and Evaluation: Use cross-validation, k-fold testing, or split testing datasets to validate model performance before deployment.

Automated Testing: Implement automated tests for preprocessing and feature extraction, ensuring these steps function consistently.

Real-time Testing Scenarios: Test with various scenarios to simulate real-world usage, like testing with dynamic lighting or hand movements.

Continuous Testing and Monitoring: After deployment, monitor the system in the production environment and test new updates before release.

7.6 TEST CASES

For your project, test cases should cover a variety of inputs, scenarios, and expected outcomes. Some examples:

1. **Basic Detection Test:** Ensure that each predefined hand sign is correctly identified.
   Input: Image or video of a specific hand sign.
   Expected Output: The model identifies the hand sign accurately.
2. **Boundary Cases:** Test how the model performs with slight variations in hand positions, like tilted angles or partial hand visibility.
   Input: Hand signs slightly angled or only partially visible.
   Expected Output: Model either correctly identifies or fails gracefully (e.g., with a low confidence score).
3. **Lighting Variation Test:** Test under different lighting conditions.
   Input: Images under low light, bright light, or with shadows.
   Expected Output: Consistent accuracy or defined limitations under extreme lighting.
4. **Multiple Hands Test:** Test the model's response to multiple hands in the frame.
   Input: Two or more hands showing signs in the frame.
   Expected Output: Detect and classify each hand's sign accurately or handle with a clear limitation message if unsupported.
5. **Performance and Latency Test:** Measure the model's response time in real-time detection scenarios.
   Input: Continuous video feed with hand signs at random intervals.
   Expected Output: Response time under a specific threshold to ensure real-time performance.

8. OUTPUT SCREENS

8.1 DETECTING HAND

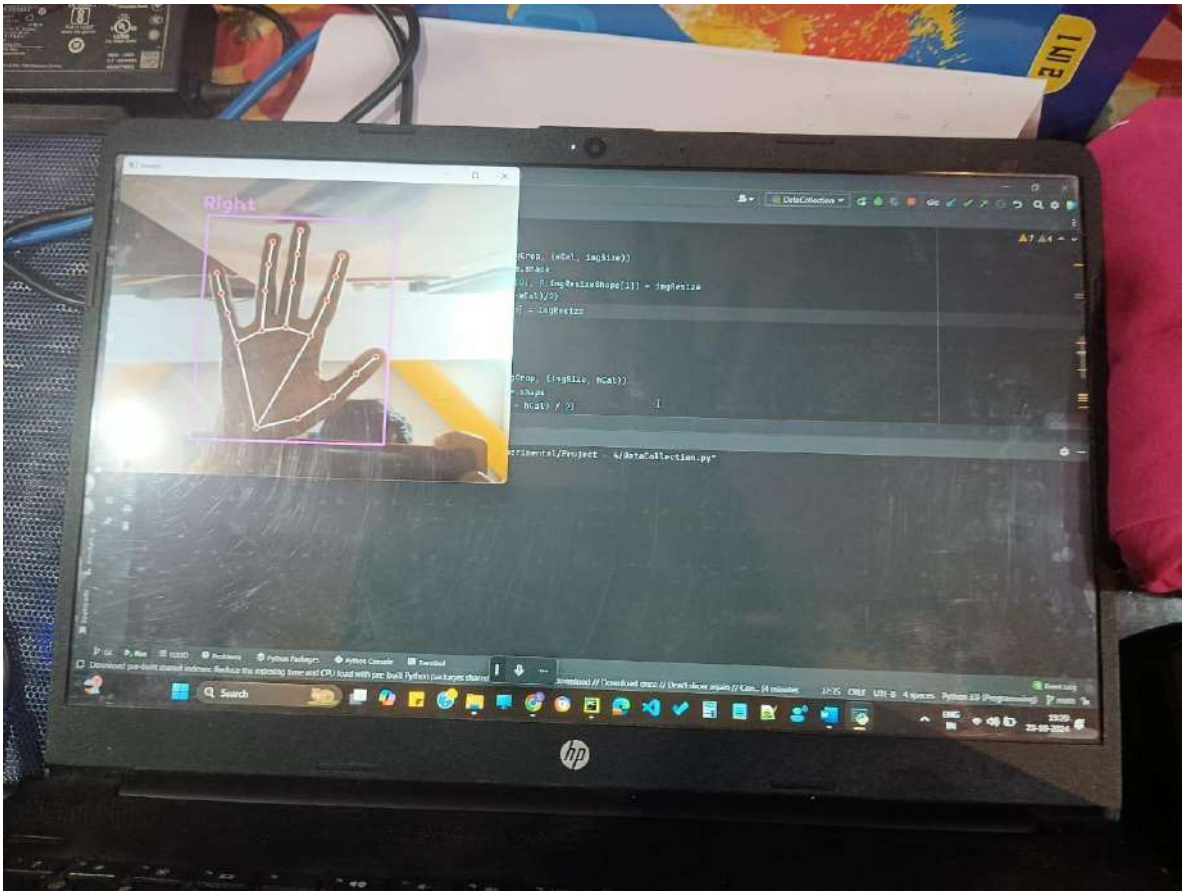

Fig 8.1 DETECTING HAND

The above shown output screen is an output of the DataCollection.py file. Here the hand is detected on a basic sense of whether it is right or left hand. This happens by the use of hand detector module in the Python library.

8.2 DETECT LETTER A

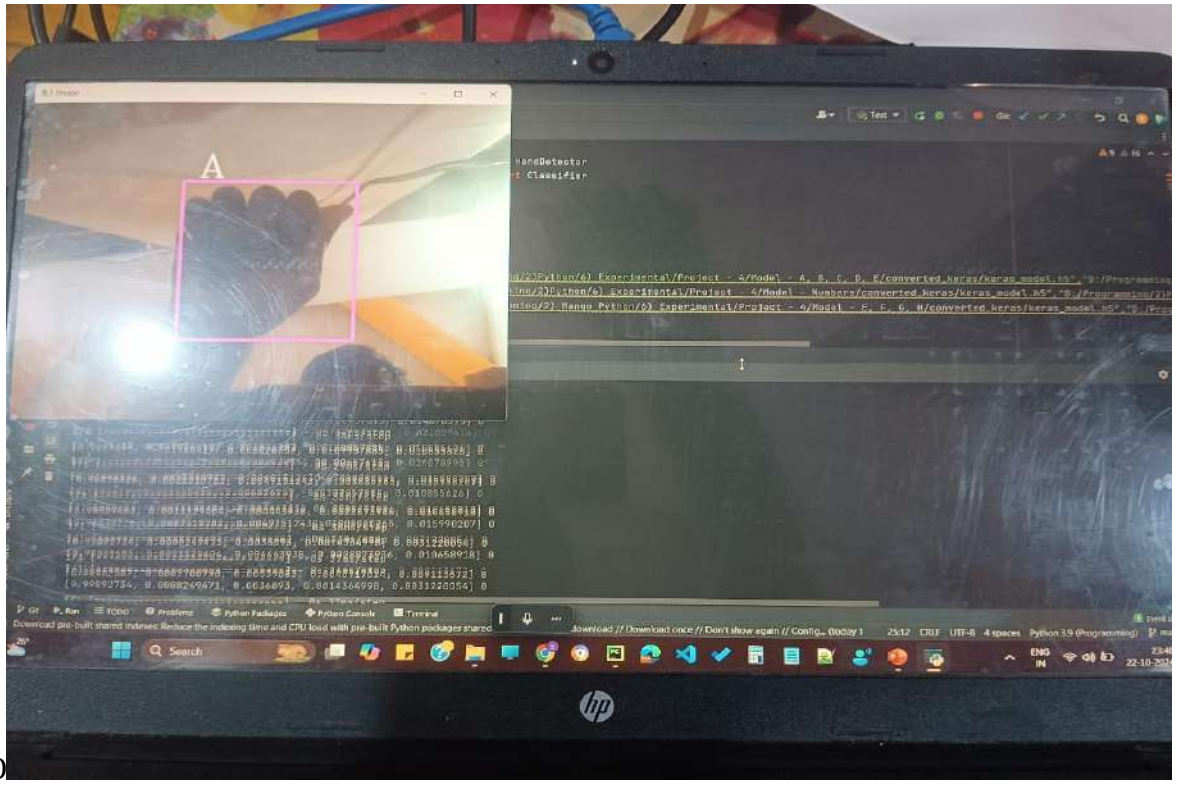

Fig 8.2 DETECT LETTER A

The above shown output screen is an output of the Test.py file. Here the hand is detected based on the training that one gives to the model to recognize that the given gesture is ‘A’. Here the model training is done by using Teachable machines website which is an open source model training website in the AI and ML model training. The underlying concepts in the website are that it uses a classification algorithm to classify the data and train accordingly. And it uses tensorflow.js for client-side processing. It finally uses PoseNet to track the pose of the hand gesture.

9. CONCLUSION & FUTURE SCOPE

9.1 CONCLUSION

The "Hand Sign Detection Using Computer Vision in Python" project successfully demonstrates the power and flexibility of computer vision technologies in creating an intuitive and interactive system for recognizing hand gestures. By leveraging Python's open-source libraries like OpenCV and TensorFlow, the system can capture live video input, process the data in real-time, and accurately detect hand signs using advanced machine learning techniques. This project highlights how touchless interfaces can significantly enhance user experiences in various applications, from accessibility tools to gaming and robotics.

The accuracy and robustness of the system depend largely on the quality of the training data and the efficiency of preprocessing techniques. Using deep learning models, specifically Convolutional Neural Networks (CNNs), the system can automatically learn and improve its performance by classifying different gestures. This project lays the groundwork for future improvements, such as expanding the gesture library, improving recognition accuracy in more complex environments, or integrating the system with external devices like smart home systems or robots.

In conclusion, this project represents a promising step toward advancing human-computer interaction through gesture-based control. The potential applications of hand sign detection are vast, and as technology continues to evolve, we can expect to see even more sophisticated and user-friendly systems. The success of this project paves the way for further research and development, making it a valuable contribution to the fields of computer vision, machine learning, and interactive system design.

9.2 FUTURE SCOPE

Hand sign detection using computer vision has a promising future across several fields, thanks to advancements in AI and deep learning. Here's a look at the future scope for this technology:

Assistive Technology:

Hand sign detection can make communication easier for people with hearing and speech impairments by translating sign language into text or speech. Enhanced detection systems could improve accessibility, allowing seamless interaction in everyday life and bridging the communication gap.

Human-Computer Interaction (HCI):

Gesture-based interfaces could enable new ways of interacting with computers, smart devices, and AR/VR systems. This has applications in gaming, education, remote work, and navigation, especially in environments where hands-free operation is valuable (e.g., healthcare or manufacturing).

Augmented Reality (AR) and Virtual Reality (VR):

Hand sign detection can enable natural gesture-based controls in AR/VR, making immersive experiences more intuitive and interactive. This has particular relevance in fields like gaming, virtual meetings, and simulations, where lifelike interactions add value.

Robotics and Automation:

Robots can be controlled via hand gestures, facilitating remote or automated operations in hazardous environments, such as in search and rescue or industrial maintenance. This could make it easier to instruct robots in complex tasks without needing a joystick or specialized controller.

Education and Training:

Hand sign detection can enhance learning platforms, particularly in language and vocational training. For example, it could help users learn sign language by providing feedback, or simulate real-life scenarios requiring hand signals, such as medical procedures or construction.

Security and Surveillance:

In security applications, hand gestures can be used to signal distress or communicate without speaking, which could be beneficial in situations where voice communication is risky. Hand sign recognition could also serve as an additional layer of biometric security.

Healthcare and Therapy:

For patients undergoing physical therapy or rehabilitation, hand sign detection can help track progress, guide exercises, and provide real-time feedback. This could also benefit people with motor skill disabilities, offering them new ways to interact with technology.

E-commerce and Retail:

With gesture recognition, customers could browse through virtual stores or try on products using hand gestures, offering a touchless, immersive shopping experience. This can be particularly valuable in AR-enabled online retail spaces.

Public Spaces and Smart Environments:

In smart cities, hand gestures could be used to control public kiosks, navigate maps, or give commands to interactive displays, reducing the need to touch shared surfaces and enhancing hygiene in crowded areas.

Personalized Smart Homes:

In home automation, hand signs can be used to control appliances, lighting, and media. For example, specific gestures could activate appliances, control music, or adjust lighting without needing a phone or remote.

10. BIBLIOGRAPHY

10.1 WEBSITES :

Hand Gesture Recognition Based on Computer Vision:

A Review of Techniques Munir Oudah.

Website:

https://doi.org/10.3390/jimaging6080073

Hand Gesture Recognition:

Published in *International Journal of Artificial Intelligence.*

Website: https://doi.org/10.5121/ijaia.2012.3412

Hand Gesture Recognition System Based in Computer :

Cooperative Neighbor Selection for Reliable Transmission

Website:

https://link.springer.com/chapter/10.1007/978-3-319-13407-9_21

10.2 REFERENCES :

[1] J Imaging. 2020 Jul 23;6(8):73. doi: 10.3390/jimaging6080073 Hand Gesture Recognition Based on Computer Vision: A Review of Techniques Munir Oudah 1, Ali Al-Naji 1,2, *, Javaan Chahl 2 Copyright and License information PMCID: PMC8321080 PMID: 34460688

[2] Hand Gesture Recognition: A Literature Review August 2012 International Journal of Artificial Intelligence & Applications 3(4):161-174 DOI:10.5121/ijaia.2012.3412

[3] Hand Gesture Recognition Based on Computer Vision: A Review of Techniques Munir Oudah 1, Ali Al-Naji 1,2, *, Javaan Chahl 2

[4] Hand Gesture Recognition System Based in Computer Vision and Machine Learning October 2013 DOI:10.1007/978-3-319-13407-9 21I.Ravi Prakash Reddy, "Trust Rate based SCooperative Neighbor Selection for Reliable Data Transmission in Mobile Adhoc Networks", International Journal of Research, ISSN No:2236- 6124, IJR Journal, Vol-VII.