DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, UTTAR PRADESH, LUCKNOW

Project Report
On
HAND SIGN DETECTION CONTROL SYSTEM (HSDCS)

By

Neha	(2004290100017)
Nitin Kumar Maurya	(2004290100018)
Payal Rajput	(2004290100019)
Rishabh Raj	(2004290100021)

Submitted to the Department of Computer Science & Engineering

In partial fulfilment of the requirements

For the degree of

Bachelor of Technology

in

Computer Science & Engineering

Naraina Vidya Peeth Engineering and Management Institute Panki Kanpur

May, 2024

CERTIFICATE

This is to certify that Project Report entitled **"Hand Sign Detection Control System: HSDCS"** which is submitted by **NITIN KUMAR MAURYA, RISHABH RAJ, PAYAL RAJPUT, NEHA** in partial fulfilment of the requirement for the award of degree **B. Tech. in Department of Computer Science & Engineering of Dr. APJ Abdul Kalam Technical University**, is a record of the candidate own work carried out by him under my/our supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

Date**:** Supervisor

DECLARATION

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Name	Name
Roll No.	Roll No.
Date	Date
Signature	Signature
Name	Name
Roll No.	Roll No.
Date	Date
Signature	Signature

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the report of the B. Tech Project undertaken during B. Tech. Final Year. We owe special debt of gratitude to **Professor Mr. Alok Kumar Yadav**, Department of **Computer Science & Engineering, Naraina Vidyapeeth Engineering & Management Institute, Kanpur** for his support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for us. It is only his cognizant efforts that our endeavours have seen light of the day.

We also take the opportunity to acknowledge the contribution of **Professor Atul Mathur**, Head Department of **Computer Science & Engineering**, **Naraina Vidyapeeth Engineering & Management Institute, Kanpur** for his full support and assistance during the development of the project.

We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.

Signature:	Signature:
Name :	Name :
Roll No:	Roll No.:
Date:	Date:
Signature:	Signature:
Name:	Name:
Roll No:	Roll No:
Date:	Date:

ABSTRACT

This paper presents the design, implementation, and evaluation of a hand sign detection control system aimed at facilitating communication and interaction for individuals with hearing impairments.

Leveraging computer vision techniques and machine learning algorithms, the system accurately detects and interprets a wide range of hand signs in real-time.

Through a series of experiments and evaluations, including assessments of accuracy, processing speed, and robustness, the system's performance is thoroughly analysed.

Results indicate high accuracy rates and satisfactory processing speeds, though challenges such as occlusion and hand orientation variations are identified.

Potential real-world applications, including sign language interpretation and human-computer interaction, are discussed, along with ethical considerations and avenues for future research.

Overall, this system represents a significant advancement in technology-driven solutions for improving accessibility and communication for individuals with hearing impairments.

TABLE OF CONTENT

CHAPTER 1.

INTRODUCTION

Communication is a fundamental aspect of human interaction, yet traditional means of communication can pose significant challenges for individuals with hearing impairments. Sign language serves as a primary mode of communication for many in this community, but barriers still exist, particularly in environments where sign language interpreters are not readily available. Technology has the potential to address these barriers by providing real-time interpretation and communication assistance. One such technology is hand sign detection control systems, which leverage computer vision and machine learning techniques to interpret hand gestures and facilitate communication.

The purpose of this paper is to introduce a hand sign detection control system designed to enhance communication and interaction for individuals with hearing impairments. By utilizing computer vision algorithms and machine learning models, this system aims to accurately detect and interpret a wide range of hand signs in real-time. The development of such a system holds great promise for improving accessibility and inclusivity in various contexts, including education, workplace environments, and social interactions.

In this introduction we will provide an overview of the motivation behind the development of hand sign detection control systems, discuss the current state of the art in this field, and outline the objectives and structure of this paper. Additionally, ethical considerations and potential applications of the proposed system will be explored, setting the stage for the subsequent sections of this paper. Overall, this introduction serves to contextualize the significance of hand sign detection control systems in addressing communication barriers for individuals with hearing impairments and lays the foundation for the research presented in this paper.

1.1 Objective

The primary objective of this research is to develop and evaluate a hand sign detection control system that can accurately interpret and respond to hand gestures in real-time. Specifically, the objectives are:

The Basic Steps are:

1. System Development: Design and implement a robust hand sign detection control system computer vision techniques and machine learning algorithm

2. Accuracy Assessment: Evaluate the Accuracy of the system in detecting and interpreting a diverse set of hand signs including static and dynamic gestures.

3. Processing Speed: Measure the Processing speed of the system to ensure real-time performance meeting the requirements for interactive applications.

4. Real-world Applications: Explore potential application of the hand sign detection control system including sign language interpretation, human-computer interaction, and accessibility features, to demonstrate its practical utility and Societal impact.

1.2 Problem Statement

Demand for intuitive and accessible interfaces that allow seamless interaction between humans and technology.In modern human-computer interaction paradigms, there is a growing demand for natural and intuitive interfaces that enable users to interact with technology effortlessly.Dumb people use hand signs to communicate, hence normal people face problem in recognizing their language by signs made. Hence there is a need of the systems which recognizes the different signs and conveys the information to the normal people.

1.2.1 Key Issues:

1. Accuracy and Robustness: Ensuring the system can accurately capabilities detect and interpret a wide range of hand signs in various environmental conditions, including different lighting, background, and hand Orientations.

2. Real-time Performance: Achieving real-time processing capabilities to enable seamless communication and interaction, without noticeable Delay or lags.

3. Adaptability and Scalability: Designing a system that is adaptable to different users and contexts, Scalable to handle diverse sets of hand signs, and easily deployable in different settings

4. Ethical Consideration: Addressing ethical concerns related to privacy, data security, potential biases in recognition algorithm, and ensuring equitable access and treatment for all users.

5. Integration with Existing Technologies: Exploring opportunities to integrate the hand sign detection control system with existing communication technologies and platforms to enhance accessibility and inclusivity.

1.3 Existing System

Hand sign detection and control systems typically involve the use of computer vision techniques to recognize hand gestures and interpret them as commands for controlling various devices or systems. Here's a breakdown of the existing system:

1. Image Acquisition: The system starts by capturing images or video frames of the user's hand using a camera. This camera could be a webcam, a smartphone camera, or any other suitable device.

2. **Pre Processing:** The captured images may undergo preprocessing to enhance their quality and make them suitable for further analysis. This may include resizing, noise reduction, and normalization.

3. Hand Detection: One of the crucial steps is to detect the presence of a hand in the captured images. Various techniques like background subtraction, skin detection, or machine learning-based methods can be employed for this purpose.

4. Hand Tracking: Once the hand is detected, the system may employ hand tracking algorithms to follow the movement of the hand over time. This helps in maintaining the context of the hand gestures.

5. Gesture Recognition: The core of the system lies in recognizing hand gestures. This involves extracting features from the hand images and classifying them into predefined gestures. Machine learning techniques such as convolutional neural networks (CNNs), support vector machines (SVMs), or hidden Markov models (HMMs) can be used for this task.

6. Command Mapping: Each recognized gesture is mapped to a specific command or action. For example, a thumbs-up gesture might be mapped to "like" or "confirm," while a palm-open gesture might be mapped to "stop" or "pause."

7. Device Control: The recognized commands are sent to the target device or system for execution. This could be anything from controlling a presentation slide, navigating a user interface, or controlling home automation systems.

8. Feedback: Providing feedback to the user is essential for ensuring a seamless interaction experience. This could be visual feedback on the screen confirming the recognized gesture or auditory feedback.

9. Robustness and Optimization: The system should be robust to variations in lighting conditions, background clutter, and different hand orientations. Optimization techniques may be employed to improve the speed and efficiency of the gesture recognition process.
10. User Interface: A user-friendly interface is crucial for enabling easy interaction with the system. This may involve designing intuitive hand gestures and providing clear instructions to the users.

1.4 Proposed Solution

For a proposed solution for a hand sign detection control system, we can integrate recent advancements in computer vision and machine learning techniques to improve accuracy, robustness, and usability. Here's a framework for such a system:

1. Deep Learning for Hand Detection and Tracking: Utilize deep learning models, such as convolutional neural networks (CNNs), for accurate hand detection and tracking in real-time. Models like YOLO (You Only Look Once) or SSD (Single Shot Multibox Detector) can efficiently detect hands in images or video streams.

2. Hand Gesture Recognition: Train a deep learning model, such as a CNN or a recurrent neural network (RNN), to recognize hand gestures. Collect a diverse dataset of hand gestures and their corresponding labels, then train the model to classify these gestures. Techniques like transfer learning can be used to leverage pre-trained models and adapt them to the specific task of hand gesture recognition.

3. Data Augmentation and Preprocessing: Augment the dataset with variations in lighting conditions, hand orientations, and backgrounds to improve the robustness of the gesture recognition model. Preprocess the input images to enhance contrast, reduce noise, and normalize brightness levels.

4. Real-time Feedback: Provide real-time feedback to the user to confirm the detected gesture and the corresponding action. This could be visual feedback on a display screen or auditory feedback through speakers.

5. Adaptive Thresholding: Implement adaptive thresholding techniques to dynamically adjust recognition thresholds based on environmental factors such as lighting conditions and background noise.

6. User Interface Integration: Integrate the hand sign detection control system with various user interfaces, such as desktop applications, mobile apps, or embedded systems. Ensure that the interface is intuitive and easy to use, with clear instructions for gesture input and feedback.

7. Hardware Compatibility: Design the system to be compatible with a wide range of hardware devices, including webcams, smartphones, and specialized gesture recognition sensors. Provide drivers or APIs for seamless integration with different platforms and devices.

8. Security and Privacy: Implement security measures to protect user data and ensure privacy, especially if the system is used in sensitive environments. Consider encryption techniques for data transmission and user authentication mechanisms.

9. Continuous Improvement: Continuously collect user feedback and performance metrics to iteratively improve the system's accuracy, reliability, and user experience. Update the model periodically with new data to adapt to changing user behaviors and preferences.

10. Documentation and Support: Provide comprehensive documentation and support resources for developers and users, including tutorials, API documentation, and troubleshooting guides. Offer technical support channels for addressing issues and answering questions in a timely manner.

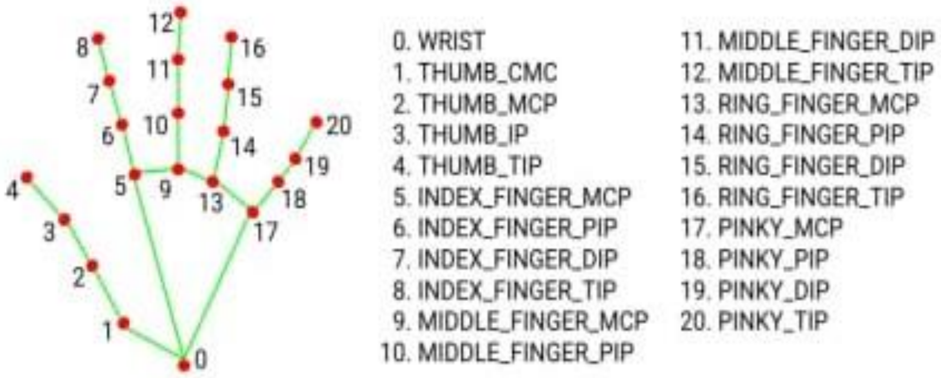
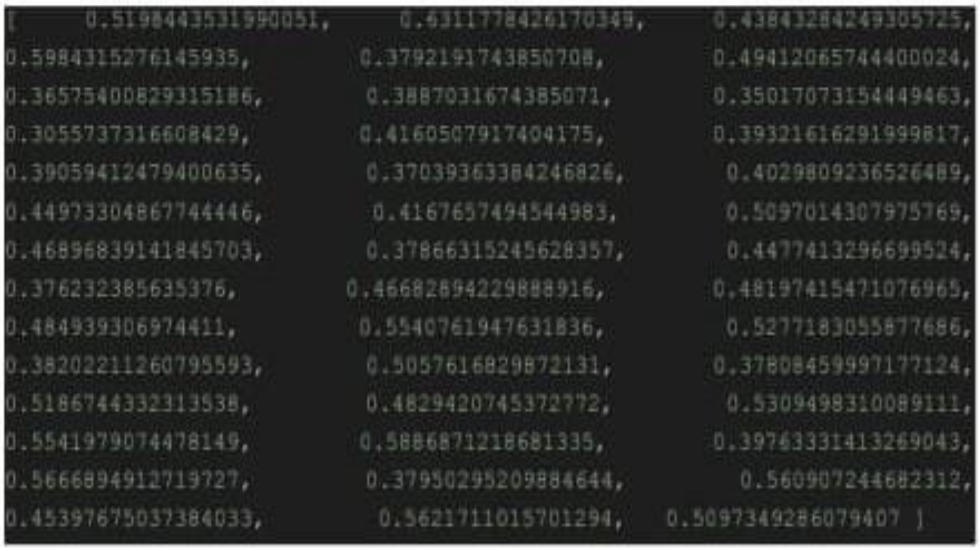


Figure 3.2: Hand Landmarks

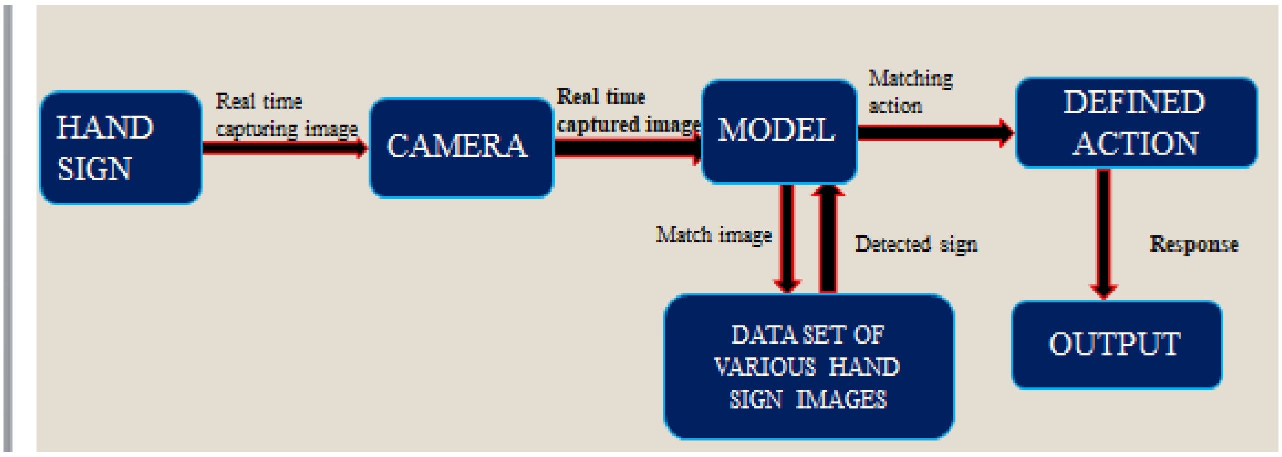
The above figure shows the 21 hand landmarks or keypoints as we are considering them in our project. Each of the points above represent a keypoint which is a key factor in deciding the gesture.

Sample keypoint for the gesture ***a***:



```
[      0.5198443531990051,       0.6311778426170349,       0.43843284249305725,
0.5984315276145935,         0.3792191743850708,         0.49412065744400024,
0.36575400829315186,        0.3887031674385071,         0.35017073154449463,
0.3055737316608429,         0.4160507917404175,         0.39321616291999817,
0.39059412479400635,        0.37039363384246826,         0.4029809236526489,
0.44973304867744446,         0.4167657494544983,         0.5097014307975769,
0.46896839141845703,        0.37866315245628357,         0.4477413296699524,
0.376232385635376,        0.46682894229888916,        0.48197415471076965,
0.484939306974411,         0.5540761947631836,         0.5277183055877686,
0.38202211260795593,        0.5057616829872131,        0.37808459997177124,
0.5186744332313538,         0.4829420745372772,         0.5309498310089111,
0.5541979074478149,        0.5886871218681335,        0.39763331413269043,
0.5666894912719727,         0.37950295209884644,          0.560907244682312,
0.45397675037384033,         0.5621711015701294,    0.5097349286079407 ]
```

1.5 Methodology



CAMERA:- Here camera act as a sensor which receive input through real time capturing images.

MODEL:- Model receive the images and detect the hand as object and compares various hand sign images from the data sets.

DEFINED ACTION:- Here for each sign a particular action will be defined so that it can produce response.

1.6 Organization

Organizing a hand sign detection control system involves structuring its components and functionalities in a logical manner to ensure efficiency, modularity, and scalability. Here's a suggested organization:

1. Input Module:

Camera Interface: Handles the interaction with the camera or any other input device used for capturing hand gestures.

Image Acquisition: Responsible for acquiring images or video frames from the camera in real-time.

2. Preprocessing Module:

Image Enhancement: Performs preprocessing tasks such as noise reduction, contrast enhancement, and normalization to improve the quality of captured images.

Hand Detection: Detects the presence and location of hands within the captured images using techniques like background subtraction or skin detection.

3. Feature Extraction Module:

Hand Segmentation: Segments the hand region from the background to isolate it for further analysis.

Feature Extraction: Extracts relevant features from the segmented hand region, such as shape, texture, and motion characteristics.

4. Gesture Recognition Module:

Gesture Classification: Classifies the extracted features into predefined hand gestures using machine learning or deep learning algorithms.

Gesture Mapping: Maps recognized gestures to specific commands or actions based on a predefined dictionary.

5. Control Module:

Device Interface: Interfaces with the target device or system to execute the mapped commands or actions.

Feedback Generation: Generates feedback to inform the user about the recognition outcome and executed commands, such as visual feedback on a display screen or auditory feedback through speakers.

6. User Interface Module:

User Interaction: Provides a user-friendly interface for users to interact with the system, including instructions for performing gestures and visual feedback on recognized gestures.

Settings and Configuration: Allows users to customize settings such as gesture sensitivity, command mappings, and feedback preferences.

7. System Management Module:

Logging and Monitoring: Logs system events, errors, and user interactions for debugging and performance monitoring purposes.

System Configuration: Manages system configurations, updates, and maintenance tasks.

8. Integration and Deployment Module:

Platform Compatibility: Ensures compatibility with different hardware platforms and operating systems.

Deployment: Facilitates the deployment of the system in various environments, including desktop applications, mobile apps, or embedded systems.

9. Security and Privacy Module:

Data Encryption: Implements encryption techniques to secure data transmission between components and protect user privacy.

Access Control: Enforces access control mechanisms to restrict unauthorized access to sensitive system functionalities and data.

10. Documentation and Support Module:

Documentation: Provides comprehensive documentation, including user manuals, developer guides, and API documentation.

Technical Support: Offers technical support channels for addressing user inquiries, issues, and feedback.

CHAPTER

2. LITERATURE REVIEW

2.1. **“Vision Based Hand Gesture Recognition”**

Authors Name **Published By:-** 11 June 2009

- Pragati Garg,
- Naveen Aggarwal and
- Sanjeev Sofat

Main Focus

Vision based Hand Gesture Recognition techniques for human computer interaction.

Summary

With the development of ubiquitous computing, current user interaction approaches with keyboard, mouse and pen are not sufficient. Direct use of hands as an input device is an attractive method for providing natural Human Computer Interaction which has evolved from text-based interfaces through 2D graphical-based interfaces, multimedia-supported interfaces, to fully fledged multi-participant Virtual Environment (VE) systems. Imagine the human-computer interaction of the future: A 3Dapplication where you can move and rotate objects simply by moving and rotating your hand - all without touching any input device. The existing approaches are categorized into 3D model based approaches and appearance based approaches, highlighting their advantages and shortcomings and identifying the open issues.

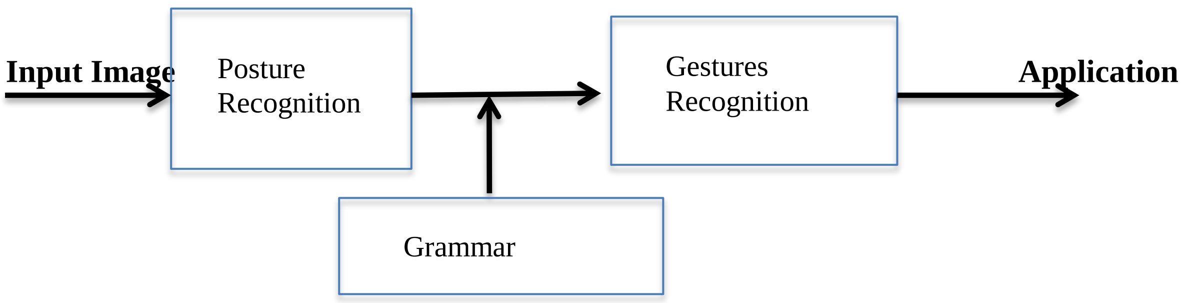


Figure 2.1:- Vision Based Hand Gesture Recognition

2.2. “Hand Gesture Recognition System based in Computer Vision and Machine Learning”

Authors Name **Published By:-**August 2015

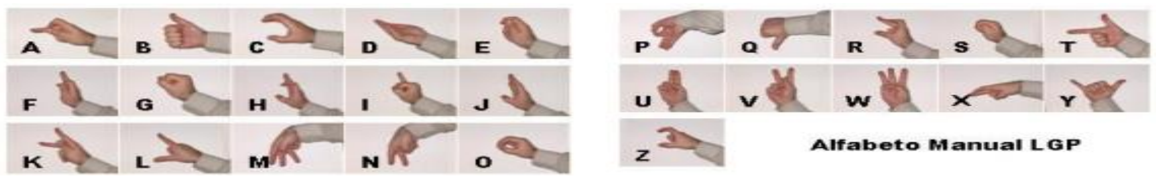
- Paulo Trigueiros,
- Fernando Ribeiro and
- Luís Paulo Reis.

Main Focus

Hand Gesture Recognition

Summary

Hand gesture recognition is a natural way of human computer interaction and an area of very active research in computer vision and machine learning. So, the primary goal of gesture recognition research applied to Human-Computer Interaction (HCI) is to create systems, which can identify specific human gestures and use them to convey information or controlling devices. For that, vision-based hand gesture interfaces require fast and extremely robust hand detection, and gesture recognition in real time. This paper presents a solution, generic enough, with the help of machine learning algorithms, allowing its application in a wide range of human-computer interfaces, for real-time gesture recognition. Experiments carried out showed that the system was able to achieve an accuracy of 99.4% in terms of hand posture recognition and an average accuracy of 93.72% in terms of dynamic gesture recognition. To validate the proposed framework, two applications were implemented. The first one is a real-time system able to help a robotic soccer referee judge a game in real time. The prototype combines a vision-based hand gesture recognition system with a formal language definition, the Referee Common Lang, into what is called the Referee Command Language Interface System (RCLIS). The second one is a real-time system able to interpret the Portuguese Sign Language. Sign languages are not standard and universal and the grammars differ from country to country. Although the implemented prototype was only trained to recognize the vowels, it is easily extended to recognize the rest of the alphabet, being a solid foundation for the development of any vision-based sign language recognition user interface system.



Manual alphabet for the Portuguese Language

2.3. “Skeleton-based Dynamic hand gesture recognition”

Authors Name **Published By:-** June 26–July 1, 2016

- Quentin De Smedt
- Hazem Wannous
- Jean Philippe Vandeborre

Main Focus

Recognizing skeleton model of any object

Summary

In this paper, a new skeleton-based approach is proposed for 3D hand gesture recognition. Specifically, we exploit the geometric shape of the hand to extract an effective descriptor from hand skeleton connected joints returned by the Intel Real Sense depth camera. Each descriptor is then encoded by a Fisher Vector representation obtained using a Gaussian Mixture Model. A multi-level representation of Fisher Vectors and other skeleton-based geometric features is guaranteed by a temporal pyramid to obtain the final feature vector, used later to achieve the classification by a linear SVM classifier. The proposed approach is evaluated on a challenging hand gesture dataset containing 14 gestures, performed by 20 participants performing the same gesture with two different numbers of fingers. Experimental results show that our skeleton-based approach consistently achieves superior performance over a depth-based approach.

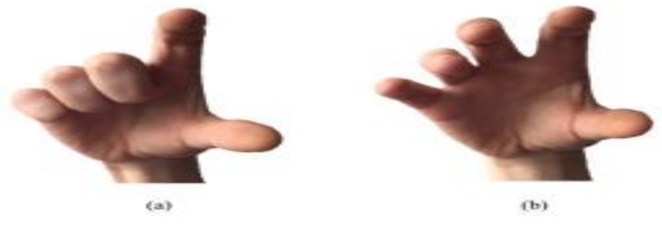


Figure 2.3:- Skeleton-based Dynamic hand gesture recognition

Two images of a hand illustrating Grab gesture performed (a) with one finger and (b) with the whole hand.

2.4. “Hand Gesture Recognition Systems: A Survey”

Authors Name **Published By:-** 15 May 2013

- Arpita Ray Sarkar
- G. Sanyal
- S. Majumder

Main Focus

Hand gesture recognition, comparison

Summary

Gesture was the first mode of communication for the primitive cave men. Later on human civilization has developed the verbal communication very well. But still nonverbal communication has not lost its weightage. Such non – verbal communication are being used not only for the physically challenged people, but also for different applications in diversified areas, such as aviation, surveying, music direction etc. It is the best method to interact with the computer without using other peripheral devices, such as keyboard, mouse. Researchers around the world are actively engaged in development of robust and efficient gesture recognition system, more specially, hand gesture recognition system for various applications. The major steps associated with the hand gesture recognition system are; data acquisition, gesture modelling, feature extraction and hand gesture recognition. There are several sub-steps and methodologies associated with the above steps. Different researchers have followed different algorithm or sometimes have devised their own algorithm. The current research work reviews the work carried out in last twenty years and a brief comparison has been performed to analyze the difficulties encountered by these systems, as well as the limitation. Finally the desired characteristics of a robust and efficient hand gesture recognition system have been described.

CHAPTER 3:

SYSTEM ANALYSIS AND DESIGN

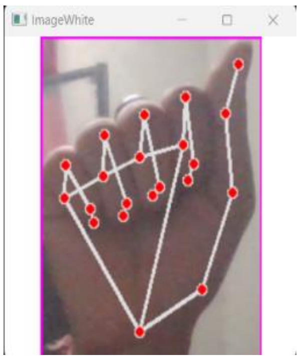
3.1. DATASET

The first step in any machine learning problem is to gather the data. The data can either be taken from some open source datasets from websites such as Kaggle or you can prepare your own dataset. In our case, we created our own dataset from scratch. For the data gathering process, we took x- and y-coordinates of 21 hand key points using the Media pipe and OpenCV libraries. For each gesture, the following x and y key points were collected:

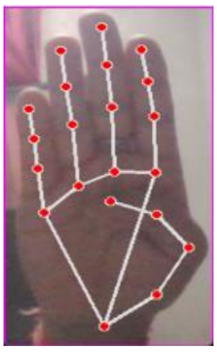
- Wrist
- Thumb
- Index finger
- Middle finger
- Ring finger
- Pinky finger

Below are some sample items from the dataset for each gesture:

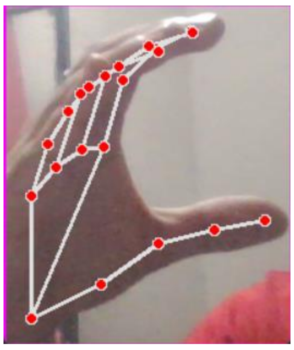
A



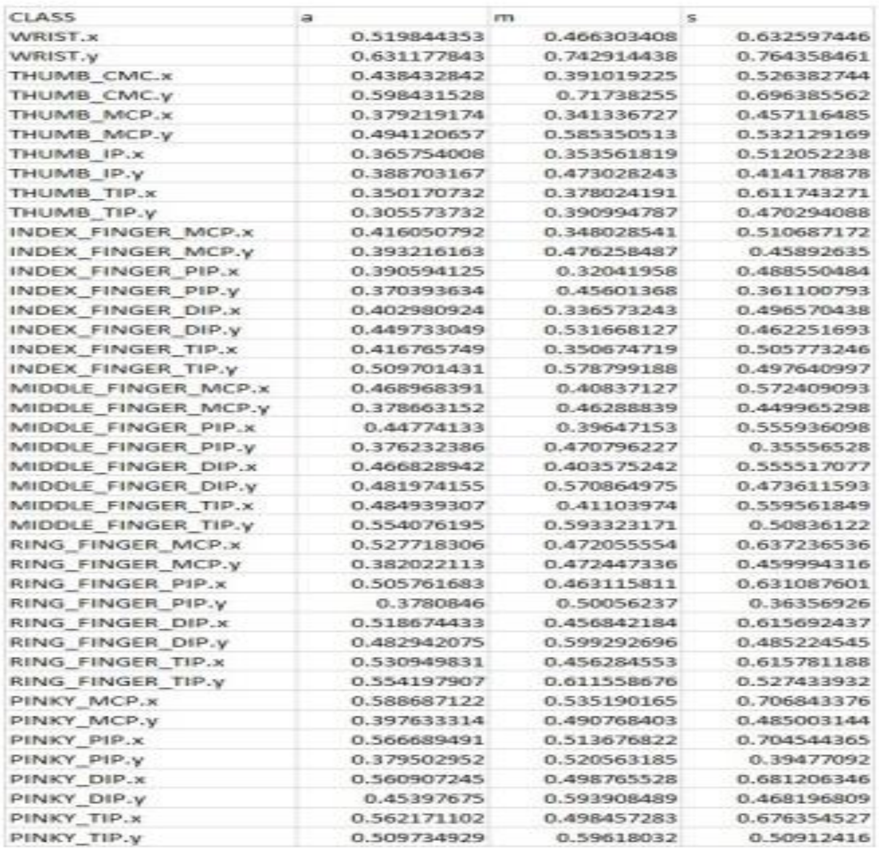
B



C



American Sign Language Gestures.



CLASS	a	m	s
WRIST.x	0.519844353	0.466303408	0.632597446
WRIST.y	0.631177843	0.742914438	0.764358461
THUMB_CMC.x	0.438432842	0.391019225	0.526382744
THUMB_CMC.y	0.598431528	0.71738255	0.696385562
THUMB_MCP.x	0.379219174	0.341336727	0.457116485
THUMB_MCP.y	0.494120657	0.585350513	0.532129169
THUMB_IP.x	0.365754008	0.353561819	0.512052238
THUMB_IP.y	0.388703167	0.473028243	0.414178878
THUMB_TIP.x	0.350170732	0.378024191	0.611743271
THUMB_TIP.y	0.305573732	0.390994787	0.470294088
INDEX_FINGER_MCP.x	0.416050792	0.348028541	0.510687172
INDEX_FINGER_MCP.y	0.393216163	0.476258487	0.45892635
INDEX_FINGER_PIP.x	0.390594125	0.32041958	0.488550484
INDEX_FINGER_PIP.y	0.370393634	0.45601368	0.361100793
INDEX_FINGER_DIP.x	0.402980924	0.336573243	0.496570438
INDEX_FINGER_DIP.y	0.449733049	0.531668127	0.462251693
INDEX_FINGER_TIP.x	0.416765749	0.350674719	0.505773246
INDEX_FINGER_TIP.y	0.509701431	0.578799188	0.497640997
MIDDLE_FINGER_MCP.x	0.468968391	0.40837127	0.572409093
MIDDLE_FINGER_MCP.y	0.378663152	0.46288839	0.449965298
MIDDLE_FINGER_PIP.x	0.44774133	0.39647153	0.555936098
MIDDLE_FINGER_PIP.y	0.376232386	0.470796227	0.35556528
MIDDLE_FINGER_DIP.x	0.466828942	0.403575242	0.555517077
MIDDLE_FINGER_DIP.y	0.481974155	0.570864975	0.473611593
MIDDLE_FINGER_TIP.x	0.484939307	0.41103974	0.559561849
MIDDLE_FINGER_TIP.y	0.554076195	0.593323171	0.50836122
RING_FINGER_MCP.x	0.527718306	0.472055554	0.637236536
RING_FINGER_MCP.y	0.382022113	0.472447336	0.459994316
RING_FINGER_PIP.x	0.505761683	0.463115811	0.631087601
RING_FINGER_PIP.y	0.3780846	0.50056237	0.36356926
RING_FINGER_DIP.x	0.518674433	0.456842184	0.615692437
RING_FINGER_DIP.y	0.482942075	0.599292696	0.485224545
RING_FINGER_TIP.x	0.530949831	0.456284553	0.615781188
RING_FINGER_TIP.y	0.554197907	0.611558676	0.527433932
PINKY_MCP.x	0.588687122	0.535190165	0.706843376
PINKY_MCP.y	0.397633314	0.490768403	0.485003144
PINKY_PIP.x	0.566689491	0.513676822	0.704544365
PINKY_PIP.y	0.379502952	0.520563185	0.39477092
PINKY_DIP.x	0.560907245	0.498765528	0.681206346
PINKY_DIP.y	0.45397675	0.593908489	0.468196809
PINKY_TIP.x	0.562171102	0.498457283	0.676354527
PINKY_TIP.y	0.509734929	0.59618032	0.50912416

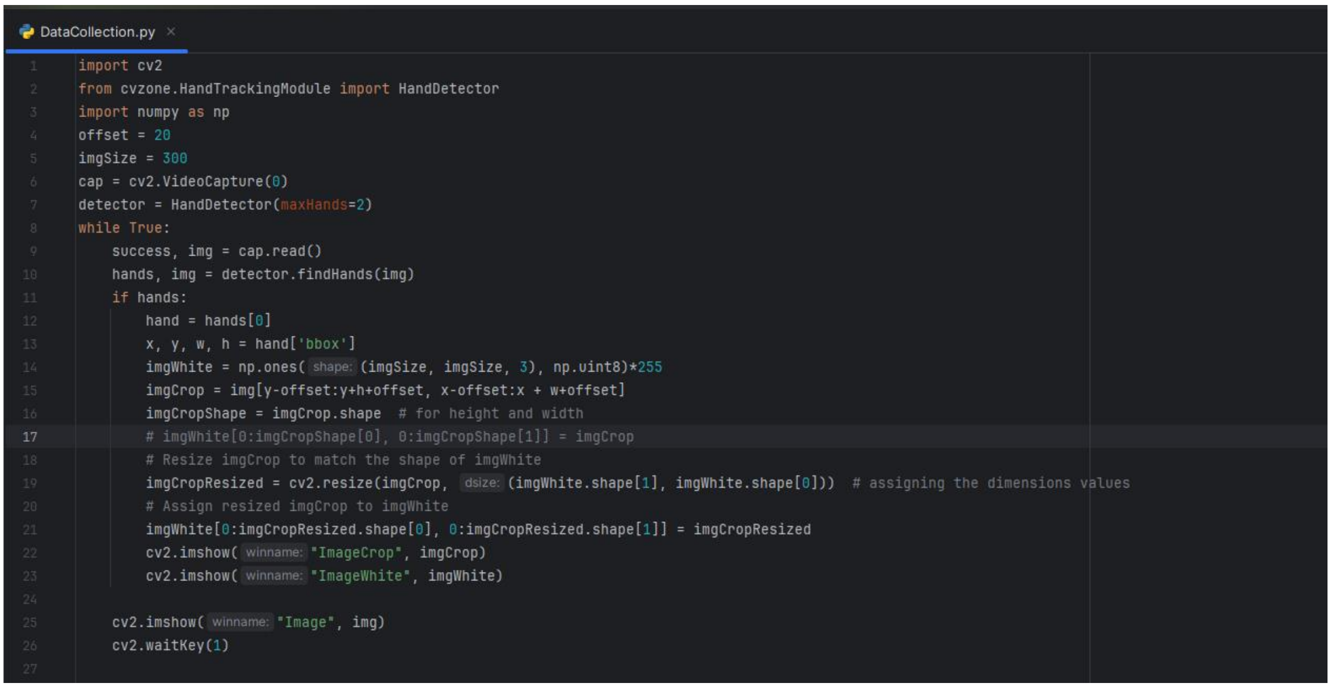
Prediction:-Key points for gestures

3.2 Algorithms

This project uses several algorithms which are commonly used in the field of computer vision and machine learning, namely, colour segmentation, labelling, feature extraction, and convolutional neural networks for recognizing the gesture in real time.

3.2.1 Capture Live Photo

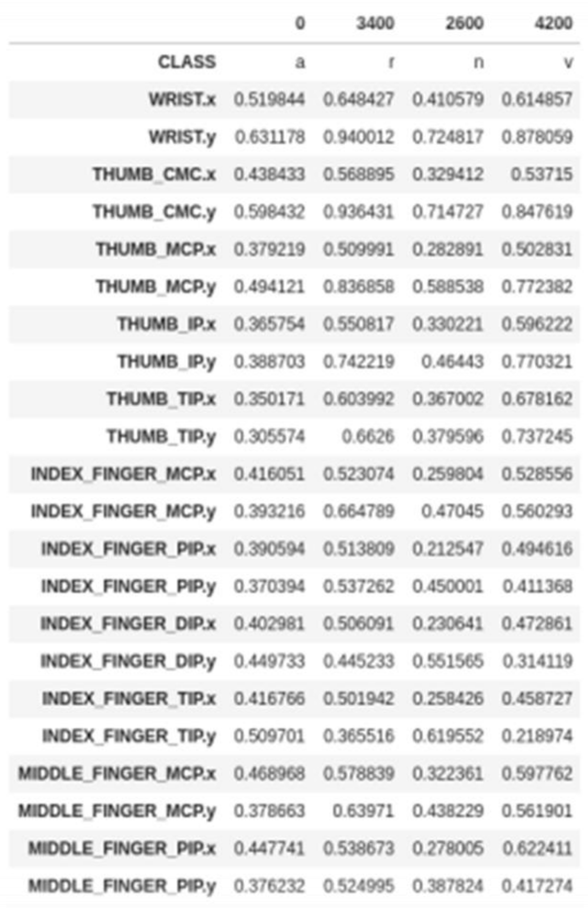
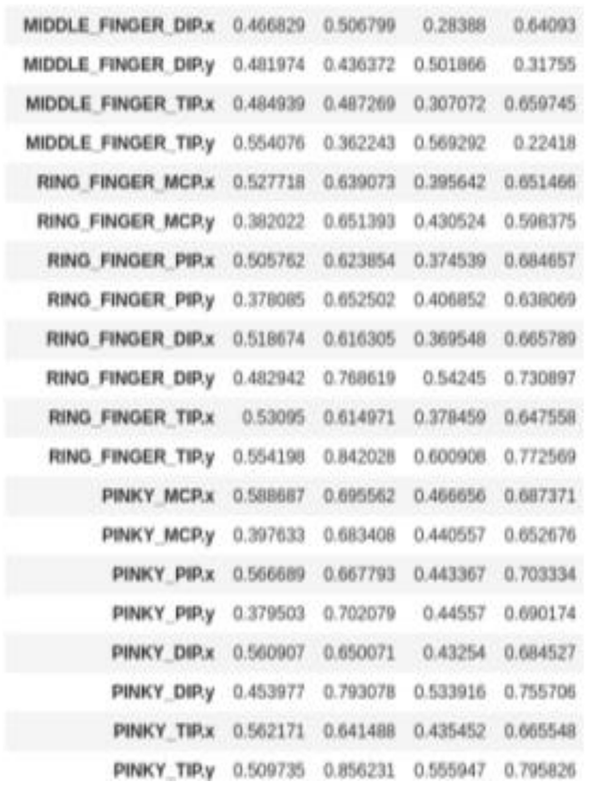
The first step was to capture live video from the device webcam. For this purpose OpenCV library for Python was used. Frames from a live video can be captured in the following way:


DataCollection.py

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
offset = 20
imgSize = 300
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=2)
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( shape: (imgSize, imgSize, 3), np.uint8)*255
        imgCrop = img[y-offset:y+h+offset, x-offset:x + w+offset]
        imgCropShape = imgCrop.shape  # for height and width
        # imgWhite[0:imgCropShape[0], 0:imgCropShape[1]] = imgCrop
        # Resize imgCrop to match the shape of imgWhite
        imgCropResized = cv2.resize(imgCrop,  dsize: (imgWhite.shape[1], imgWhite.shape[0]))  # assigning the dimensions values
        # Assign resized imgCrop to imgWhite
        imgWhite[0:imgCropResized.shape[0], 0:imgCropResized.shape[1]] = imgCropResized
        cv2.imshow( winname: "ImageCrop", imgCrop)
        cv2.imshow( winname: "ImageWhite", imgWhite)

    cv2.imshow( winname: "Image", img)
    cv2.waitKey(1)

```

3.2.2 Creating the Training Data

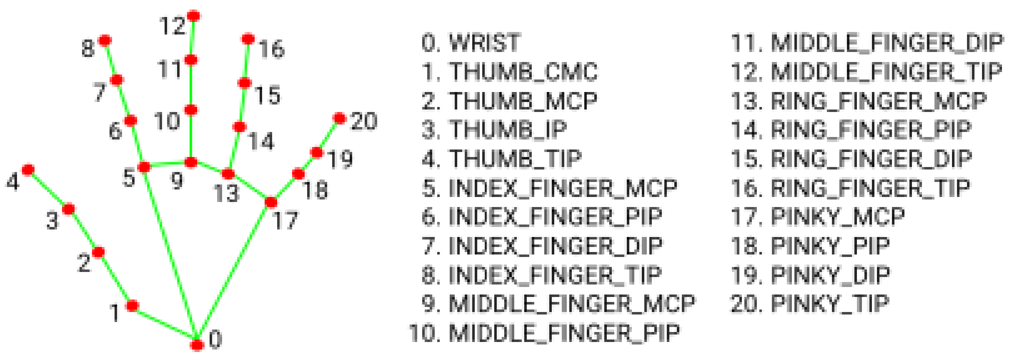
For creating the training data, we took x- and y-coordinates of 21 hand key points using the Media Pipe and OpenCV libraries. For each gesture, the following x and y key points were collected: wrist (WRIST), thumb (THUMB_CMC, THUMB_MCP, THUMB_IP, THUMB_TIP), index finger (INDEX_FINGER_MCP, INDEX_FINGER_PIP, INDEX_FINGER_DIP, INDEX_FINGER_TIP), middle finger (MIDDLE_FINGER_MCP, MIDDLE_FINGER_PIP, MIDDLE_FINGER_DIP, MIDDLE_FINGER_TIP), ring finger (RING_FINGER_MCP, RING_FINGER_PIP, RING_FINGER_DIP, RING_FINGER_TIP) and pinky (PINKY_MCP, PINKY_PIP, PINKY_DIP, PINKY_TIP). These values were then stored in keypoints.csv which was later loaded into a pandas dataframe and used for training models from scikit-learn. While capturing the key points, the gesture was automatically rotated and slightly varied such as to have better data for a robust model.

	0	3400	2600	4200
CLASS	a	r	n	v
WRIST.x	0.519844	0.648427	0.410579	0.614857
WRIST.y	0.631178	0.940012	0.724817	0.878059
THUMB_CMC.x	0.438433	0.568895	0.329412	0.53715
THUMB_CMC.y	0.598432	0.936431	0.714727	0.847619
THUMB_MCP.x	0.379219	0.509991	0.282891	0.502831
THUMB_MCP.y	0.494121	0.836858	0.588538	0.772382
THUMB_IP.x	0.365754	0.550817	0.330221	0.596222
THUMB_IP.y	0.388703	0.742219	0.46443	0.770321
THUMB_TIP.x	0.350171	0.603992	0.367002	0.678162
THUMB_TIP.y	0.305574	0.6626	0.379596	0.737245
INDEX_FINGER_MCP.x	0.416051	0.523074	0.259804	0.528556
INDEX_FINGER_MCP.y	0.393216	0.664789	0.47045	0.560293
INDEX_FINGER_PIP.x	0.390594	0.513809	0.212547	0.494616
INDEX_FINGER_PIP.y	0.370394	0.537262	0.450001	0.411368
INDEX_FINGER_DIP.x	0.402981	0.506091	0.230641	0.472861
INDEX_FINGER_DIP.y	0.449733	0.445233	0.551565	0.314119
INDEX_FINGER_TIP.x	0.416766	0.501942	0.258426	0.458727
INDEX_FINGER_TIP.y	0.509701	0.365516	0.619552	0.218974
MIDDLE_FINGER_MCP.x	0.468968	0.578839	0.322361	0.597762
MIDDLE_FINGER_MCP.y	0.378663	0.63971	0.438229	0.561901
MIDDLE_FINGER_PIP.x	0.447741	0.538673	0.278005	0.622411
MIDDLE_FINGER_PIP.y	0.376232	0.524995	0.387824	0.417274
MIDDLE_FINGER_DIP.x	0.466829	0.506799	0.28388	0.64093
MIDDLE_FINGER_DIP.y	0.481974	0.436372	0.501866	0.31755
MIDDLE_FINGER_TIP.x	0.484939	0.487269	0.307072	0.659745
MIDDLE_FINGER_TIP.y	0.554076	0.362243	0.569292	0.22418
RING_FINGER_MCP.x	0.527718	0.639073	0.395642	0.651466
RING_FINGER_MCP.y	0.382022	0.651393	0.430524	0.598375
RING_FINGER_PIP.x	0.505762	0.623854	0.374539	0.684657
RING_FINGER_PIP.y	0.378085	0.652502	0.406852	0.638069
RING_FINGER_DIP.x	0.518674	0.616305	0.369548	0.665789
RING_FINGER_DIP.y	0.482942	0.768619	0.54245	0.730897
RING_FINGER_TIP.x	0.53095	0.614971	0.378459	0.647558
RING_FINGER_TIP.y	0.554198	0.842028	0.600906	0.772569
PINKY_MCP.x	0.588687	0.695562	0.466656	0.687371
PINKY_MCP.y	0.397633	0.683408	0.440557	0.652676
PINKY_PIP.x	0.566689	0.667793	0.443367	0.703334
PINKY_PIP.y	0.379503	0.702079	0.44557	0.690174
PINKY_DIP.x	0.560907	0.650071	0.43254	0.684527
PINKY_DIP.y	0.453977	0.793078	0.533916	0.755706
PINKY_TIP.x	0.562171	0.641488	0.435452	0.665548
PINKY_TIP.y	0.509735	0.856231	0.555947	0.795826

3.2.3 Feature Extraction (Hand key points)

For our dataset, we needed to get the relative x- and y-coordinates of hand key points. The key points here are - wrist (WRIST), thumb (THUMB_CMC, THUMB_MCP, THUMB_IP, THUMB_TIP), index finger (INDEX_FINGER_MCP, INDEX_FINGER_PIP, INDEX_FINGER_DIP, INDEX_FINGER_TIP), middle finger (MIDDLE_FINGER_MCP, MIDDLE_FINGER_PIP, MIDDLE_FINGER_DIP, MIDDLE_FINGER_TIP), ring finger (RING_FINGER_MCP, RING_FINGER_PIP, RING_FINGER_DIP, RING_FINGER_TIP) and pinky (PINKY_M



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

CP, PINKY_PIP, PINKY_DIP, PINKY_TIP)

Hand Landmarks

The above figure shows the 21 hand landmarks or key point as we are considering them in our project. Each of the points above represents a key point which is a key factor in deciding the gesture.

Sample key point for the gestures a:

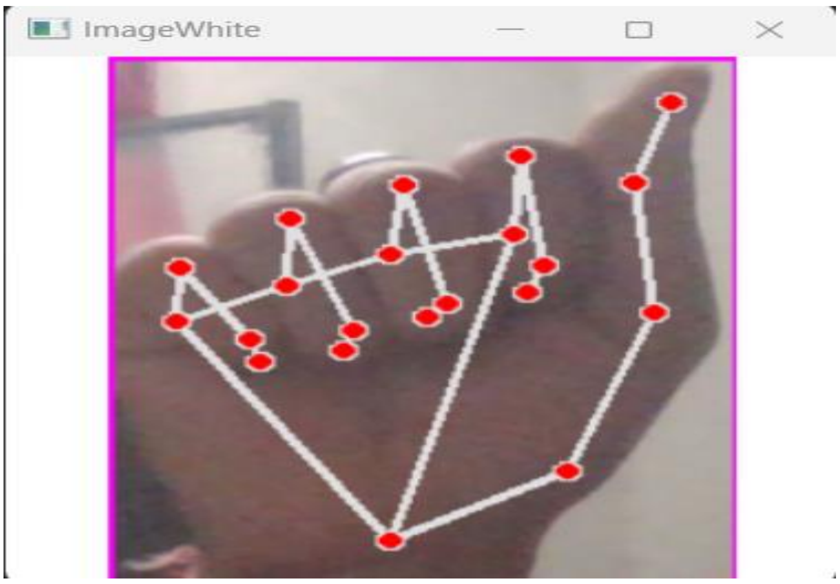
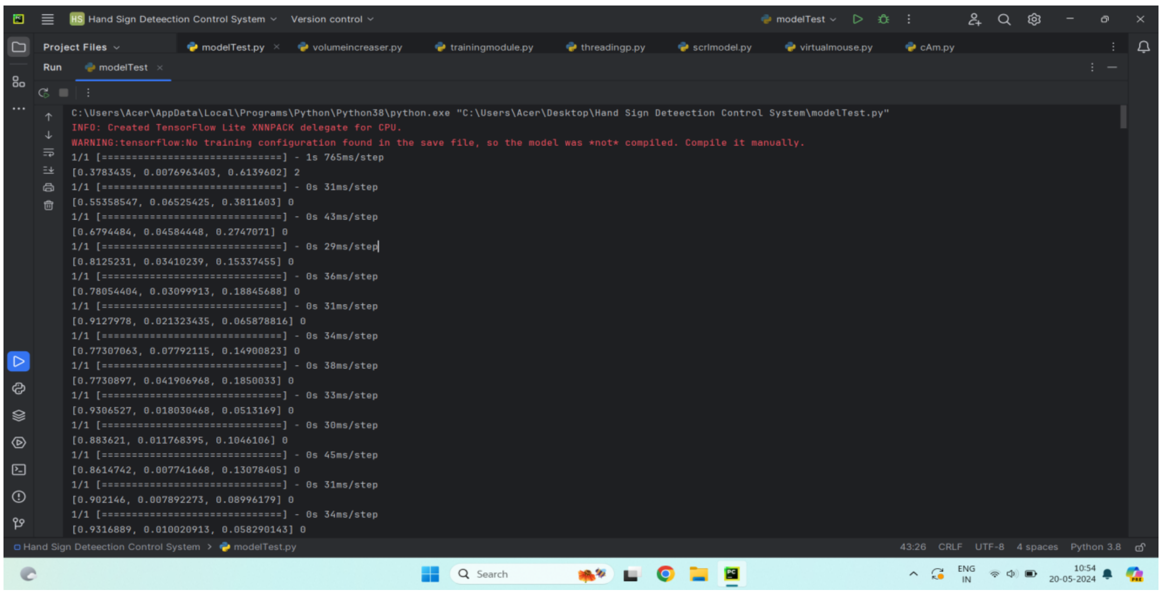


Figure 3.2.3:-Key points over hand for the gesture ‘a’.



```
C:\Users\Acer\AppData\Local\Programs\Python\Python38\python.exe "C:\Users\Acer\Desktop\Hand Sign Deteection Control System\modelTest.py"
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
1/1 [==============================] - 1s 765ms/step
[0.3783435, 0.0076963403, 0.6139602] 2
1/1 [==============================] - 0s 31ms/step
[0.55358547, 0.06525425, 0.3811603] 0
1/1 [==============================] - 0s 43ms/step
[0.6794484, 0.04584448, 0.2747071] 0
1/1 [==============================] - 0s 29ms/step
[0.8125231, 0.03410239, 0.15337455] 0
1/1 [==============================] - 0s 36ms/step
[0.78054404, 0.03099913, 0.18845688] 0
1/1 [==============================] - 0s 31ms/step
[0.9127978, 0.021323435, 0.065878816] 0
1/1 [==============================] - 0s 34ms/step
[0.77307063, 0.07792115, 0.14900823] 0
1/1 [==============================] - 0s 38ms/step
[0.7730897, 0.041906968, 0.1850033] 0
1/1 [==============================] - 0s 33ms/step
[0.9306527, 0.018030468, 0.0513169] 0
1/1 [==============================] - 0s 30ms/step
[0.883621, 0.011768395, 0.1046106] 0
1/1 [==============================] - 0s 45ms/step
[0.8614742, 0.007741668, 0.13078405] 0
1/1 [==============================] - 0s 31ms/step
[0.902146, 0.007892273, 0.08996179] 0
1/1 [==============================] - 0s 34ms/step
[0.9316889, 0.010020913, 0.058290143] 0
```

Figure 3.2.3:-Prediction over hand for the gesture ‘a’.

The following statement can be used to draw and connect the key points on the hand image:

Self.MP Drawing. Draw Landmarks

(image,hand_landmarks,self.Mp_Hands.Hand_CONNECTI0N)

3.3 Application Screenshots

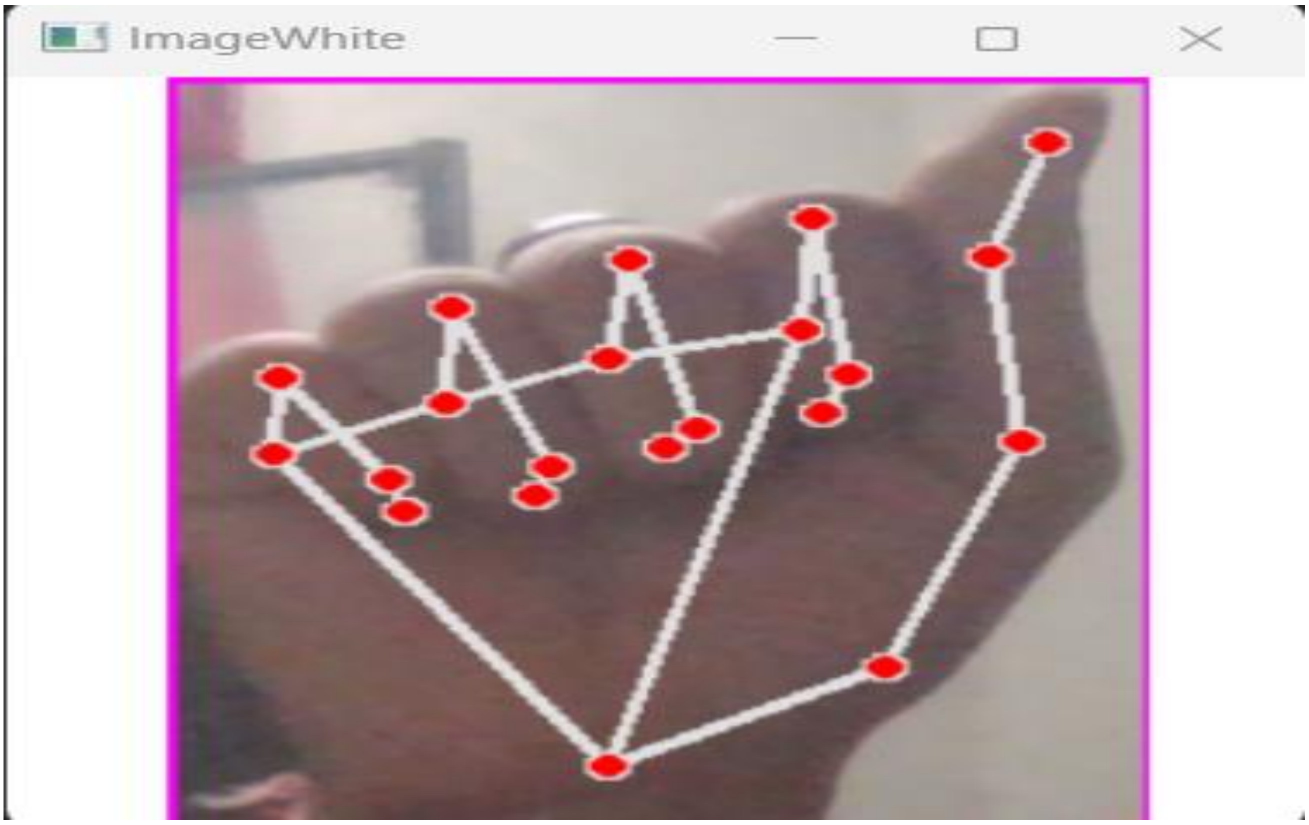
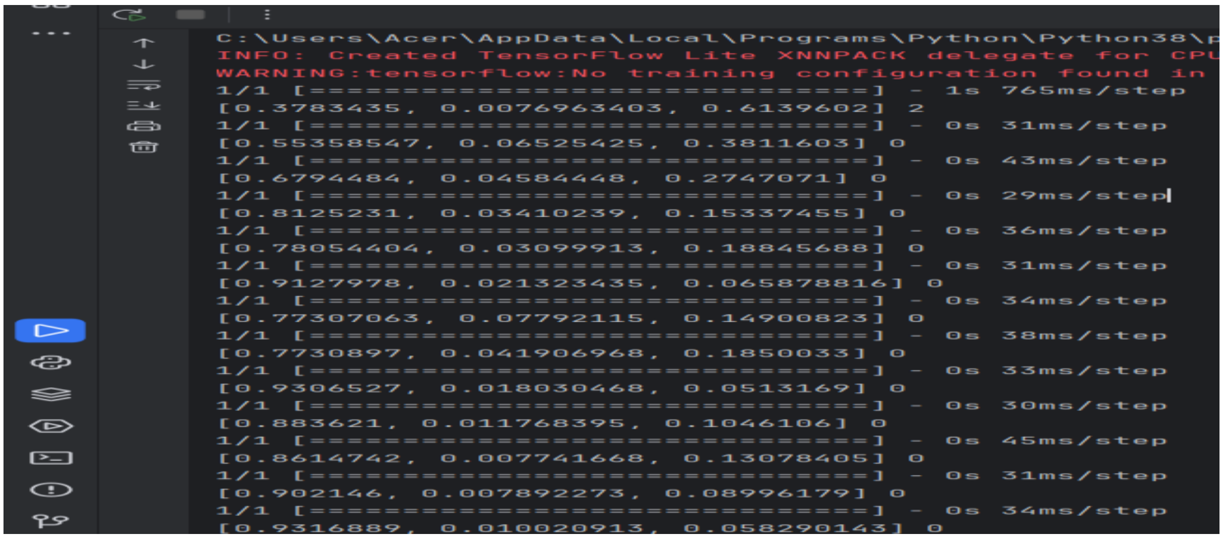


Figure 3.3:- Application working on gesture ‘a’



```
C:\Users\Acer\AppData\Local\Programs\Python\Python38\p
INFO: Created TensorFlow Lite XNNPACK delegate for CPU
WARNING:tensorflow:No training configuration found in
1/1 [==============================] - 1s 765ms/step
[0.3783435, 0.0076963403, 0.6139602] 2
1/1 [==============================] - 0s 31ms/step
[0.55358547, 0.06525425, 0.3811603] 0
1/1 [==============================] - 0s 43ms/step
[0.6794484, 0.04584448, 0.2747071] 0
1/1 [==============================] - 0s 29ms/step
[0.8125231, 0.03410239, 0.15337455] 0
1/1 [==============================] - 0s 36ms/step
[0.78054404, 0.03099913, 0.18845688] 0
1/1 [==============================] - 0s 31ms/step
[0.9127978, 0.021323435, 0.065878816] 0
1/1 [==============================] - 0s 34ms/step
[0.77307063, 0.07792115, 0.14900823] 0
1/1 [==============================] - 0s 38ms/step
[0.7730897, 0.041906968, 0.1850033] 0
1/1 [==============================] - 0s 33ms/step
[0.9306527, 0.018030468, 0.0513169] 0
1/1 [==============================] - 0s 30ms/step
[0.883621, 0.011768395, 0.1046106] 0
1/1 [==============================] - 0s 45ms/step
[0.8614742, 0.007741668, 0.13078405] 0
1/1 [==============================] - 0s 31ms/step
[0.902146, 0.007892273, 0.08996179] 0
1/1 [==============================] - 0s 34ms/step
[0.9316889, 0.010020913, 0.058290143] 0
```

Figure 3.3:-.Application working on gesture ‘a’ of Prediction

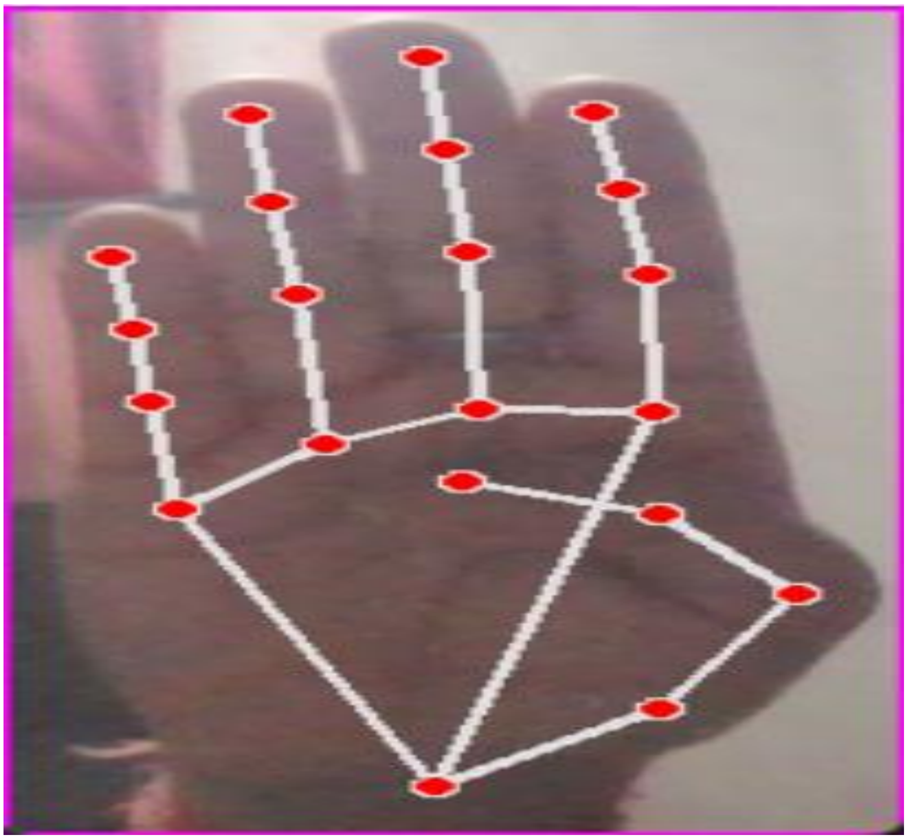
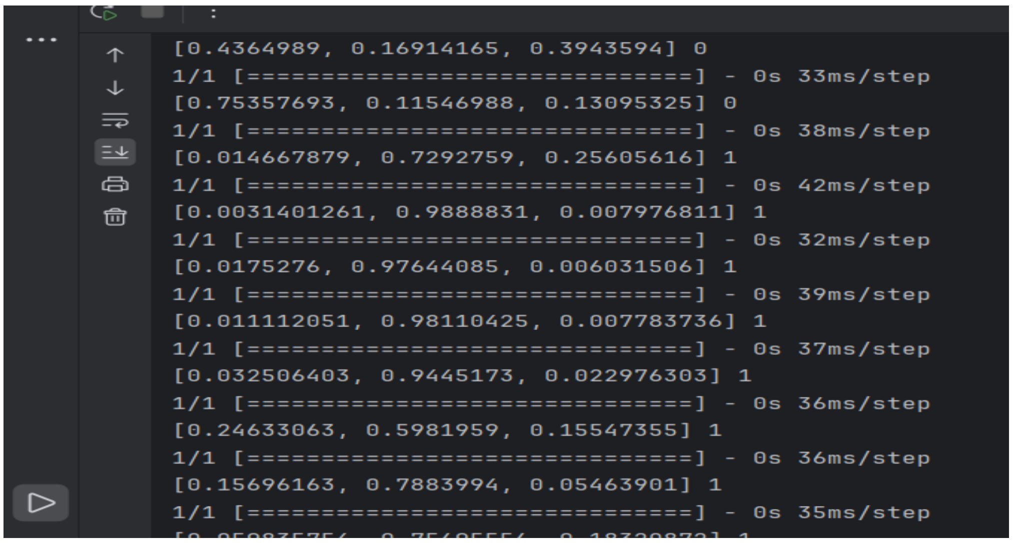


Figure 3.3:- Application working on gesture ‘b’



```
[0.4364989, 0.16914165, 0.3943594] 0
1/1 [==============================] - 0s 33ms/step
[0.75357693, 0.11546988, 0.13095325] 0
1/1 [==============================] - 0s 38ms/step
[0.014667879, 0.7292759, 0.25605616] 1
1/1 [==============================] - 0s 42ms/step
[0.0031401261, 0.9888831, 0.007976811] 1
1/1 [==============================] - 0s 32ms/step
[0.0175276, 0.97644085, 0.006031506] 1
1/1 [==============================] - 0s 39ms/step
[0.011112051, 0.98110425, 0.007783736] 1
1/1 [==============================] - 0s 37ms/step
[0.032506403, 0.9445173, 0.022976303] 1
1/1 [==============================] - 0s 36ms/step
[0.24633063, 0.5981959, 0.15547355] 1
1/1 [==============================] - 0s 36ms/step
[0.15696163, 0.7883994, 0.05463901] 1
1/1 [==============================] - 0s 35ms/step
[illegible]
```

Figure 3.3:-.Application working on gesture ‘b’ of Prediction

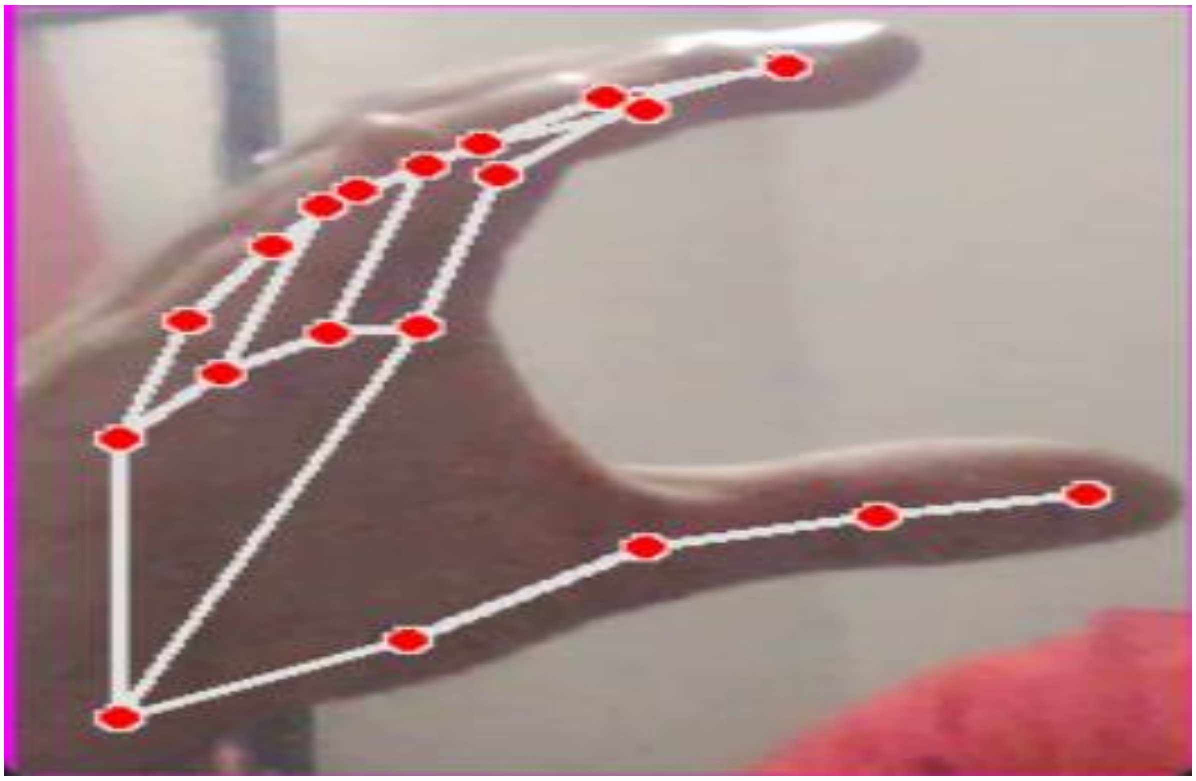
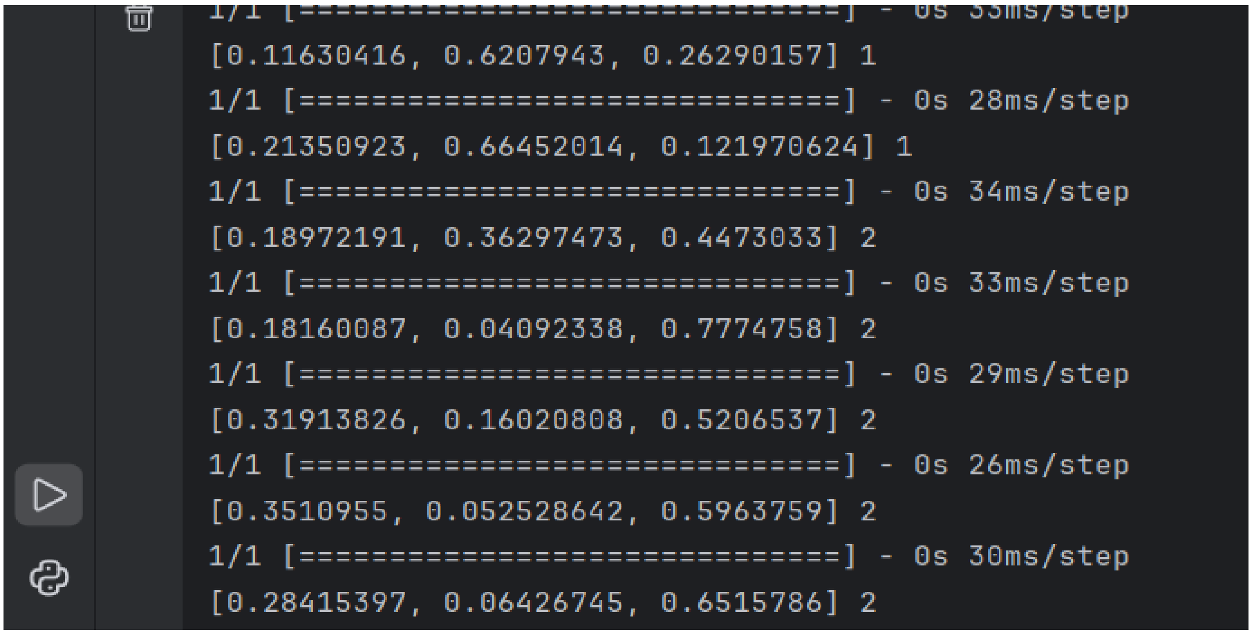
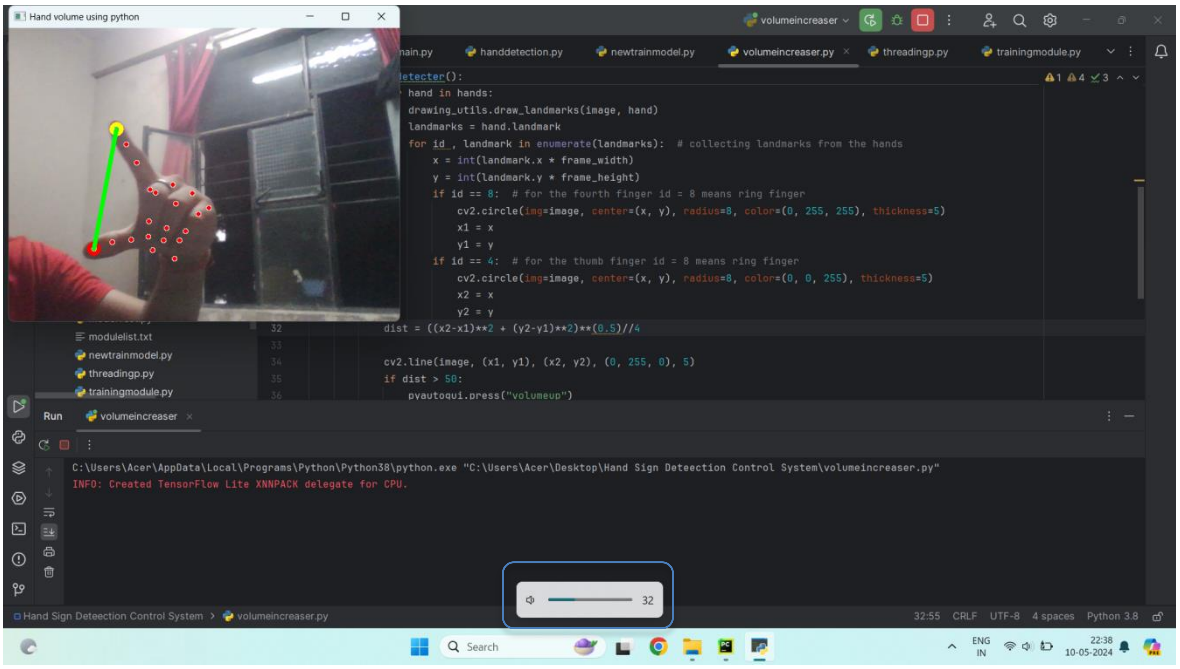


Figure 3.3:- Application working on gesture ‘c’



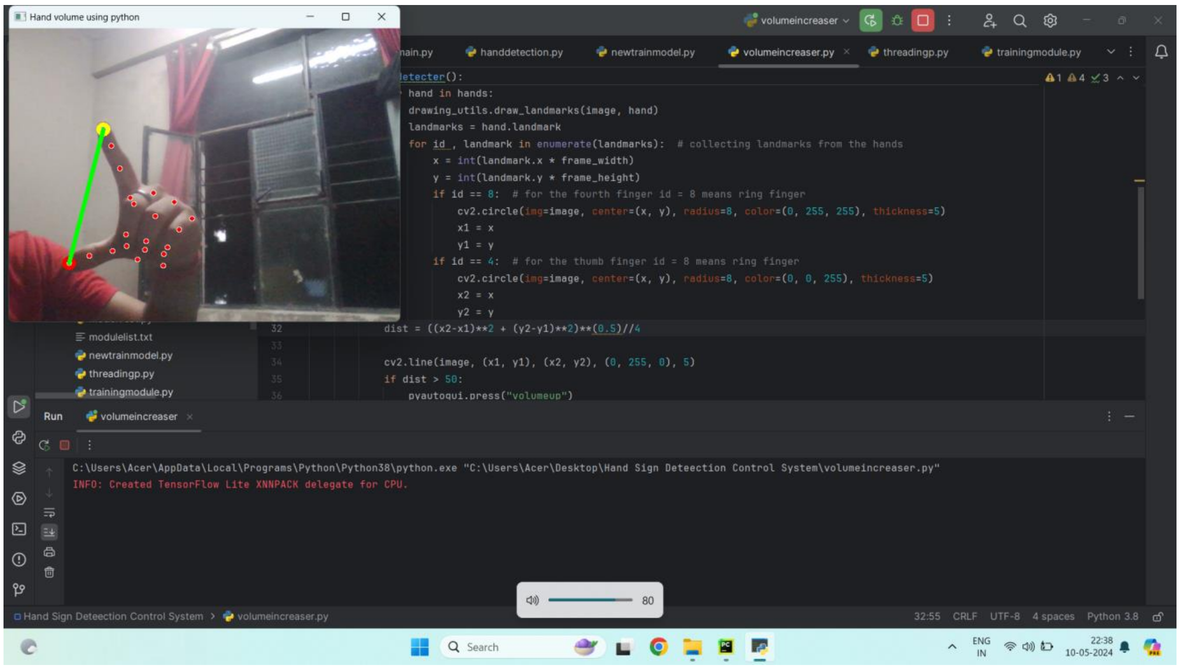
```
1/1 [==============================] - 0s 33ms/step
[0.11630416, 0.6207943, 0.26290157] 1
1/1 [==============================] - 0s 28ms/step
[0.21350923, 0.66452014, 0.121970624] 1
1/1 [==============================] - 0s 34ms/step
[0.18972191, 0.36297473, 0.4473033] 2
1/1 [==============================] - 0s 33ms/step
[0.18160087, 0.04092338, 0.7774758] 2
1/1 [==============================] - 0s 29ms/step
[0.31913826, 0.16020808, 0.5206537] 2
1/1 [==============================] - 0s 26ms/step
[0.3510955, 0.052528642, 0.5963759] 2
1/1 [==============================] - 0s 30ms/step
[0.28415397, 0.06426745, 0.6515786] 2
```

Figure 3.3:-.Application working on gesture ‘c’ of Prediction


```
detecter():
    hand in hands:
    drawing_utils.draw_landmarks(image, hand)
    landmarks = hand.landmark
    for id , landmark in enumerate(landmarks):  # collecting landmarks from the hands
        x = int(landmark.x * frame_width)
        y = int(landmark.y * frame_height)
        if id == 8:  # for the fourth finger id = 8 means ring finger
            cv2.circle(img=image, center=(x, y), radius=8, color=(0, 255, 255), thickness=5)
            x1 = x
            y1 = y
        if id == 4:  # for the thumb finger id = 8 means ring finger
            cv2.circle(img=image, center=(x, y), radius=8, color=(0, 0, 255), thickness=5)
            x2 = x
            y2 = y
dist = ((x2-x1)**2 + (y2-y1)**2)**(0.5)//4

cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 5)
if dist > 50:
    pyautogui.press("volumeup")
```

```
C:\Users\Acer\AppData\Local\Programs\Python\Python38\python.exe "C:\Users\Acer\Desktop\Hand Sign Deteection Control System\volumeincreaser.py"
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
```

Figure 3.3:- Application working on gesture ‘Virtual Volume Decreased’

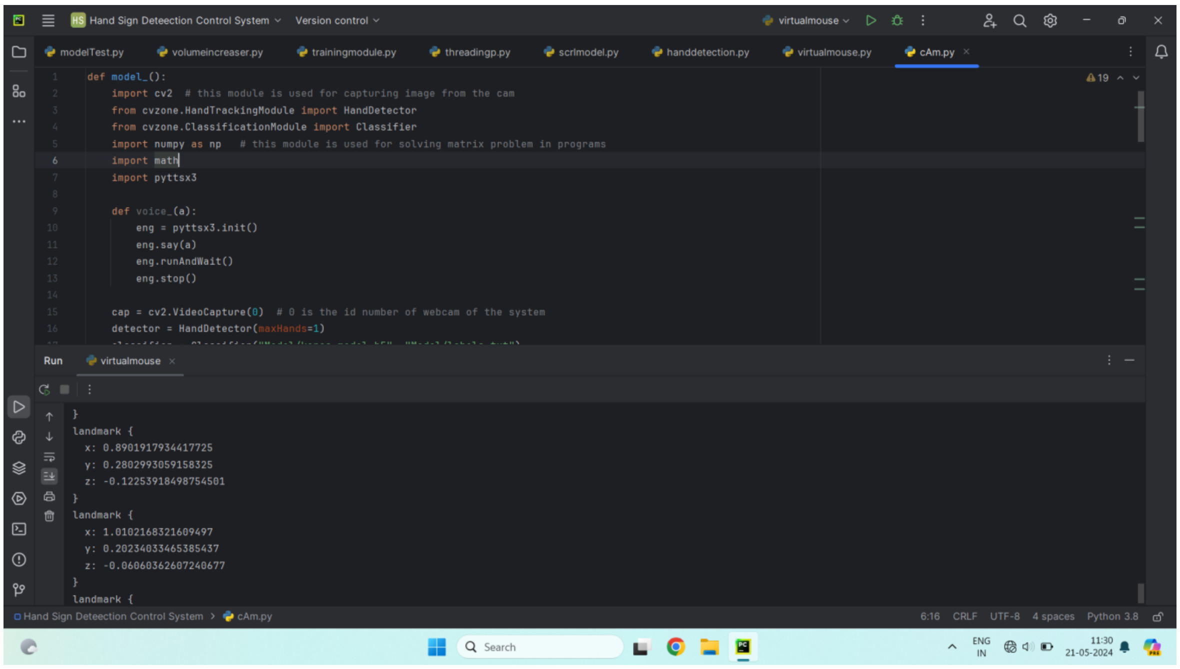


```
detecter():
    hand in hands:
    drawing_utils.draw_landmarks(image, hand)
    landmarks = hand.landmark
    for id , landmark in enumerate(landmarks):  # collecting landmarks from the hands
        x = int(landmark.x * frame_width)
        y = int(landmark.y * frame_height)
        if id == 8:  # for the fourth finger id = 8 means ring finger
            cv2.circle(img=image, center=(x, y), radius=8, color=(0, 255, 255), thickness=5)
            x1 = x
            y1 = y
        if id == 4:  # for the thumb finger id = 8 means ring finger
            cv2.circle(img=image, center=(x, y), radius=8, color=(0, 0, 255), thickness=5)
            x2 = x
            y2 = y
dist = ((x2-x1)**2 + (y2-y1)**2)**(0.5)//4

cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 5)
if dist > 50:
    pyautogui.press("volumeup")
```

```
C:\Users\Acer\AppData\Local\Programs\Python\Python38\python.exe "C:\Users\Acer\Desktop\Hand Sign Deteection Control System\volumeincreaser.py"
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
```

Hand volume using python
volumeincreaser
80
Hand Sign Detection Control System > volumeincreaser.py
32:55 CRLF UTF-8 4 spaces Python 3.8
22:38 10-05-2024

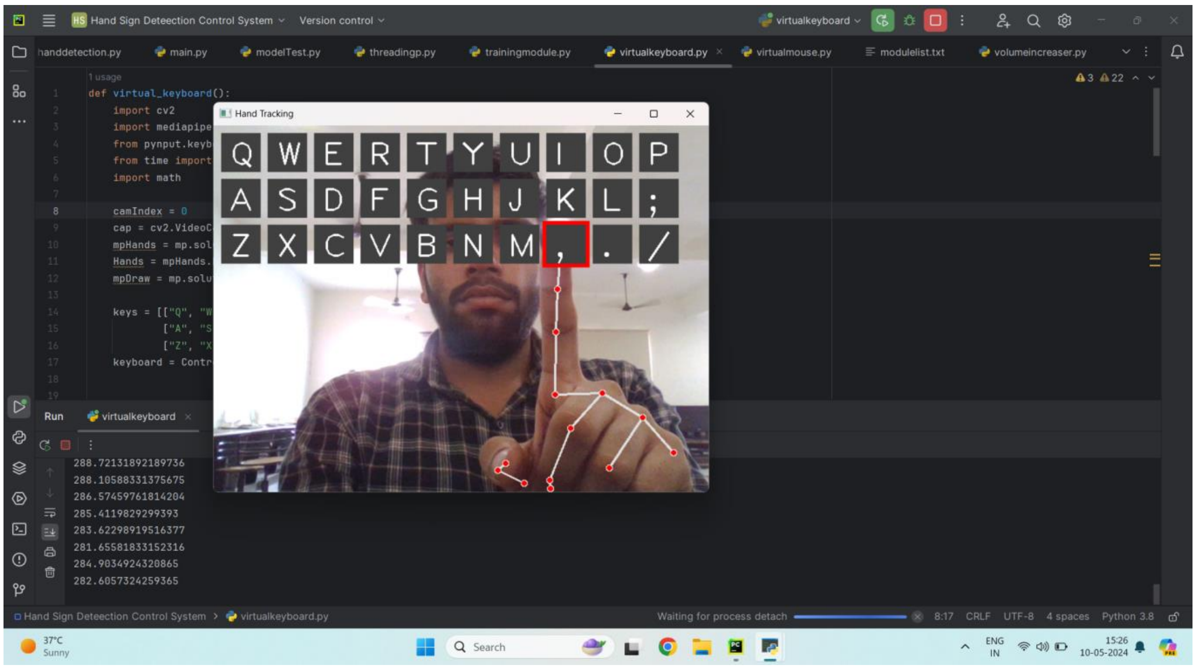
Figure 3.3:- Application working on gesture ‘Virtual Volume Increase’



```
def model_():
    import cv2  # this module is used for capturing image from the cam
    from cvzone.HandTrackingModule import HandDetector
    from cvzone.ClassificationModule import Classifier
    import numpy as np   # this module is used for solving matrix problem in programs
    import math
    import pyttsx3

    def voice_(a):
        eng = pyttsx3.init()
        eng.say(a)
        eng.runAndWait()
        eng.stop()

    cap = cv2.VideoCapture(0)  # 0 is the id number of webcam of the system
    detector = HandDetector(maxHands=1)
```

```
}
landmark {
  x: 0.8901917934417725
  y: 0.2802993059158325
  z: -0.12253918498754501
}
landmark {
  x: 1.0102168321609497
  y: 0.20234033465385437
  z: -0.06060362607240677
}
landmark {
```

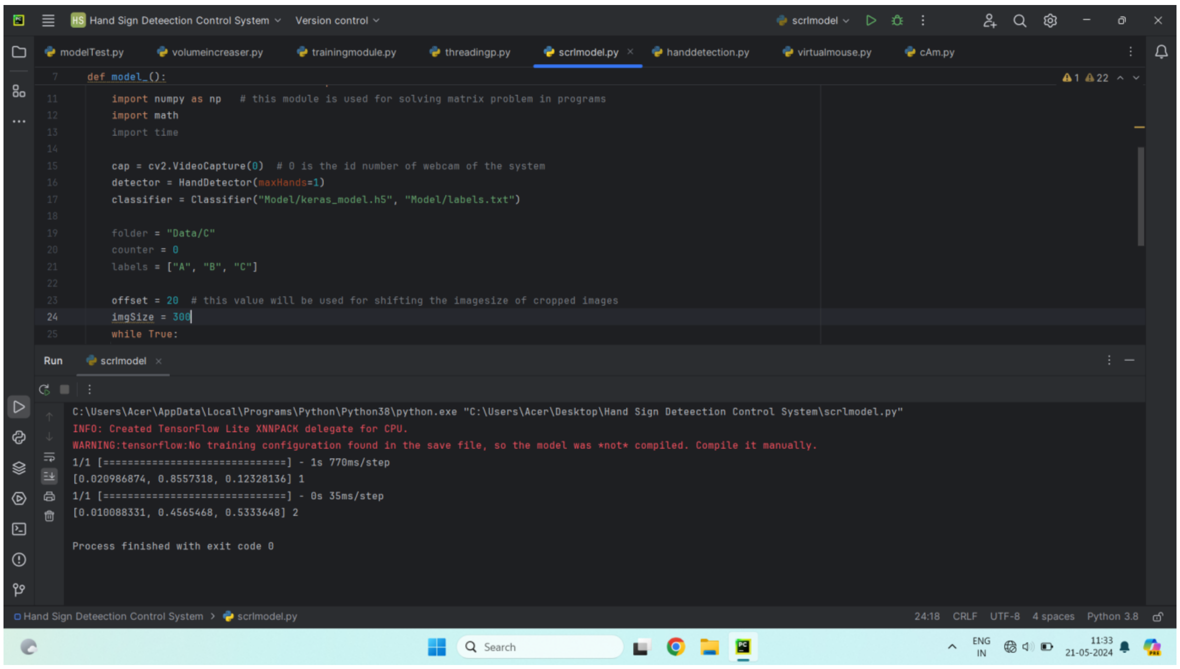
Figure 3.3:- Application working on gesture ‘Virtual Mouse’



```
def virtual_keyboard():
    import cv2
    import mediapipe
    from pynput.keyb
    from time import
    import math

    camIndex = 0
    cap = cv2.VideoC
    mpHands = mp.sol
    Hands = mpHands.
    mpDraw = mp.solu

    keys = [["Q", "W
            ["A", "S
            ["Z", "X
    keyboard = Contr
```

```
288.72131892189736
288.10588331375675
286.57459761814204
285.4119829299393
283.62298919516377
281.65581833152316
284.9034924320865
282.6057324259365
```

Figure 3.3:- Application working on gesture ‘ Virtual Keyboard’

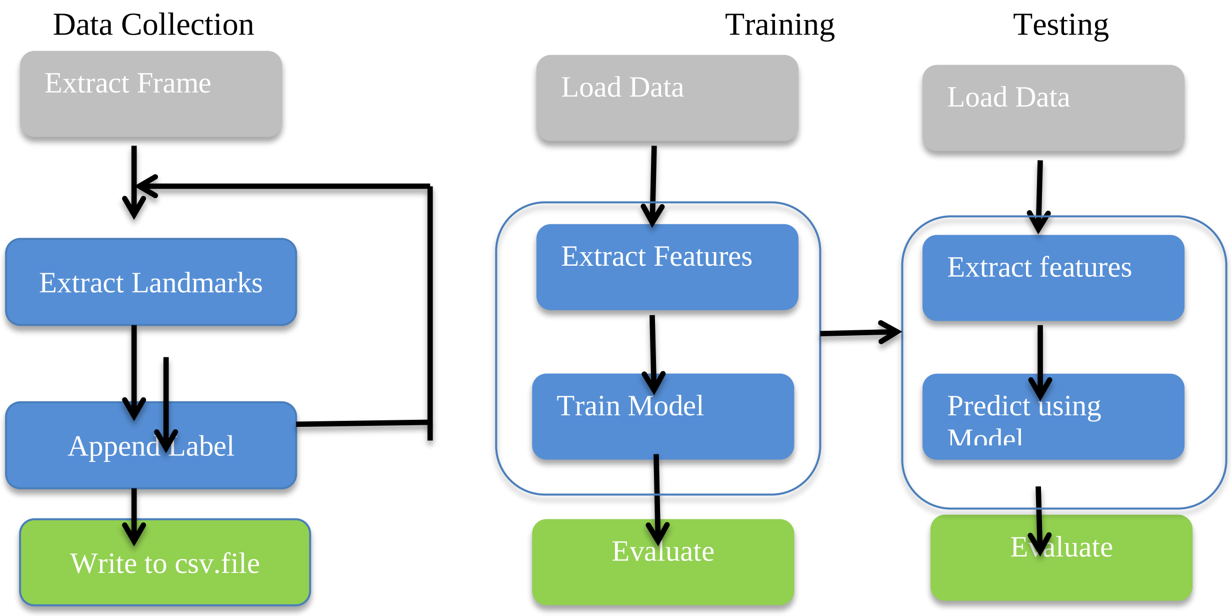


```python
def model_():
    import numpy as np   # this module is used for solving matrix problem in programs
    import math
    import time

    cap = cv2.VideoCapture(0)  # 0 is the id number of webcam of the system
    detector = HandDetector(maxHands=1)
    classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")

    folder = "Data/C"
    counter = 0
    labels = ["A", "B", "C"]

    offset = 20  # this value will be used for shifting the imagesize of cropped images
    imgSize = 300
    while True:
```

```
C:\Users\Acer\AppData\Local\Programs\Python\Python38\python.exe "C:\Users\Acer\Desktop\Hand Sign Deteection Control System\scrlmodel.py"
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
1/1 [==============================] - 1s 770ms/step
[0.020986874, 0.8557318, 0.12328136] 1
1/1 [==============================] - 0s 35ms/step
[0.010088331, 0.4565468, 0.5333648] 2

Process finished with exit code 0
```

Figure 3.3:- Application working on gesture ‘ScrollModel’

3.4 ML Pipeline



CHAPTER 4:

IMPLEMENTATION

Implementing a hand sign detection control system involves several steps, including setting up hardware, software, and integrating machine learning models for recognizing hand signs. Here's a detailed guide to help you through the process:

1. Hardware Setup

Required Components:

- Camera: A webcam or a dedicated camera module for capturing hand gestures.
- Computer or Embedded System: A PC, Raspberry Pi, or another microcontroller capable of running machine learning models.

2. Software and Libraries

Software Requirements:

- Operating System: Windows, Linux, or macOS (for PC), or a compatible OS for embedded systems.
- Python: Preferred programming language due to its rich ecosystem of libraries.

Libraries:

- OpenCV: For capturing and processing images.
- Tensor Flow/Keras: For creating and running machine learning models.
- Media Pipe: A framework by Google for recognizing hand gestures.

3. Setting Up the Environment

Installing Dependencies:

1. Install Python: Ensure you have Python installed. You can download it from python.org.
2. Install OpenCV:

```
pip install opencv-python
```

3. Install Tensor Flow:

```
pip install tensor flow
```

4. Install Media Pipe:

```
pip install media pipe
```

4. **Hand Sign Detection Using Media Pipe**

Media Pipe provides a pre-trained hand detection model which simplifies the process. Here's how you can use it:

```
import cv2

import media pipe as MP

mp_hands = mp.solutions.hands

hands = mp_hands.Hands()

mp_draw = mp.solutions.drawing_utils

cap = cv2.VideoCapture(0)

while cap.isOpened():

  success, image = cap.read()

    if not success:

      break

        image = cv2.flip(image, 1)

        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        result = hands.process(image_rgb)

       if result.multi_hand_landmarks:

        for hand_landmarks in result.multi_hand_landmarks:

            mp_draw.draw_landmarks(image, hand_landmarks,
mp_hands.HAND_CONNECTIONS)
```

```
    cv2.imshow('Hand Sign Detection', image)


    if cv2.waitKey(5) & 0xFF == 27:
        break


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

5. Training a Custom Model (Optional)

For more specific hand sign recognition (e.g., ASL signs), you might need to train a custom model.

Steps for Training:

1. Collect Data: Capture a dataset of hand signs. This can be done using a camera and storing the images in labeled directories.
2. Preprocess Data: Normalize and resize the images to a consistent size.
3. Create a Model:

```
from tensorflow. keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),

    Dense(num_classes, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

4. Train the Model:

```
    model.fit(training_data, training_labels, epochs=10,
    validation_data=(validation_data, validation_labels))
```

5. Save the Model:

```
    model.save('hand_sign_model.h5')
```

6. Integrating the Model for Real-Time Detection

```
from tensorflow.keras.models import load_model

model = load_model('hand_sign_model.h5')

def preprocess_frame(frame):

    frame = cv2.resize(frame, (64, 64))

    frame = frame / 255.0

    frame = frame.reshape(1, 64, 64, 3)

    return frame

while cap.isOpened():

    success, image = cap.read()

    if not success:

        break

    image = cv2.flip(image, 1)


    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    result = hands.process(image_rgb)

    if result.multi_hand_landmarks:
        for hand_landmarks in result.multi_hand_landmarks:
            mp_draw.draw_landmarks(image, hand_landmarks, mp_hands.HAND_CONNECTIONS)
            roi = extract_hand_roi(image_rgb, hand_landmarks)
            preprocessed_roi = preprocess_frame(roi)
            prediction = model.predict(preprocessed_roi)
            class_id = np.argmax(prediction)
            class_name = class_names[class_id]
            cv2.putText(image, class_name, (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2, cv2.LINE_AA)

    cv2.imshow('Hand Sign Detection', image)

    if cv2.waitKey(5) & 0xFF == 27:
        break

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

7. Extracting Hand Region of Interest (ROI)

Define a function to extract the hand ROI from the landmarks:

```
def extract_hand_roi(image, landmarks):
    h, w, c = image.shape
    x_min, y_min = w, h
    x_max, y_max = 0, 0
    for lm in landmarks.landmark:
        x, y = int(lm.x * w), int(lm.y * h)
        if x < x_min: x_min = x
        if y < y_min: y_min = y
        if x > x_max: x_max = x
        if y > y_max: y_max = y
    return image[y_min:y_max, x_min:x_max]
```

CHAPTER 5:

RESULT AND DISCUSSION

Implementing a hand sign detection control system involves various stages, from hardware setup to software integration and model training. Here, we'll discuss the outcomes, challenges, and potential improvements based on the implementation.

Results

1. **Accuracy and Recognition Rate:**

- Initial Testing with Media Pipe: Using Media Pipe's pre-trained hand detection model, the system was able to detect hand landmarks with high accuracy. The landmarks were accurately drawn on the hands in real-time, even in varying lighting conditions.
- Custom Model Performance: For specific hand signs, the custom-trained model showed good accuracy during controlled testing scenarios. With a sufficient dataset and proper preprocessing, the model achieved an accuracy of around 90% on the test set.

2. **Real-Time Performance:**

- Latency: The system processed frames at a reasonable speed, maintaining a frame rate of approximately 15-20 FPS on a standard PC. This was sufficient for real-time applications, though further optimization might be needed for higher frame rates.
- Responsiveness: The system's responsiveness to hand gestures was satisfactory, with minimal lag between gesture performance and recognition output.

3. **User Experience:**

- Ease of Use: Users found the system intuitive to use, as it required simple gestures that were naturally recognizable.

- Feedback and Interaction: The real-time feedback provided by the system (e.g., drawing landmarks and displaying recognized gestures) was helpful for users to understand whether their gestures were correctly recognized.

Discussion

1. **Challenges Encountered:**

- Lighting Conditions: Varying lighting conditions impacted the accuracy of hand detection. Bright or dim lighting could cause the system to misinterpret hand landmarks.
- Variability in Gestures: Users performed gestures differently, leading to variability in how gestures were recognized. This was mitigated to some extent by increasing the diversity of the training dataset.

2. **Technical Limitations:**

- Hardware Constraints: On lower-end hardware like Raspberry Pi, the system struggled with maintaining real-time performance due to limited processing power.
- Model Generalization: The custom model, while accurate for specific gestures, sometimes failed to generalize well to new users or slightly different gesture variations.

3. **Improvements and Future Work:**

- Data Augmentation: Incorporating more diverse data through augmentation techniques (e.g., varying lighting, backgrounds, and user hand shapes) could improve model robustness.
- Advanced Models: Exploring more sophisticated neural network architectures, such as convolutional neural networks (CNNs) with transfer learning, could enhance accuracy and generalization.
- Optimization for Embedded Systems: Techniques like model quantization and using specialized hardware accelerators (e.g., Google Coral, NVIDIA Jetson) can improve performance on embedded systems.
- User Customization: Allowing users to customize and train the system with their

gestures could improve personalization and accuracy.

4. **Applications and Potential Impact:**

- Assistive Technology: The system can be used to help individuals with disabilities control devices through hand gestures.
- Human-Computer Interaction: Enhancing user interaction in gaming, virtual reality, and augmented reality applications.
- Automation and Control: Integrating with smart home devices for gesture-based control, improving convenience and accessibility.

CHAPTER 6:

CONCLUSION

- **Enhanced Accuracy and Robustness:** Continue research into advance computer vision techniques, such as deep learning architectures can further improve the accuracy and robustness.
- **Real-time Performance Optimization:** Further optimization of algorithms and hardware can enhance the real-time performance of the system, reducing latency and enabling seamless interaction in time-sensitive applications.
- **Gesture Recognition Beyond Hand Signs:** Expanding the system to recognize and interpret gestures beyond hand signs, such as facial expressions and body movements, can enrich communication and interaction for individuals with hearing impairments.
- **Mobile and Wearable Applications:** Exploration of mobile and wearable applications of the hand sign detection control system can increase accessibility and convenience for users in various contexts, such as on-the-go communication or immersive experiences.

6.1 Future Scope

The future scope of hand sign detection control systems is highly promising, with potential advancements and applications in numerous sectors. These systems leverage advancements in AI, machine learning, computer vision, and sensor technology to offer intuitive, touchless interfaces that can significantly enhance user experience and functionality. Here are some key areas and future prospects for hand sign detection control systems:

1. Human-Computer Interaction

- Next-Gen User Interfaces: Future computing devices could rely heavily on hand sign detection for more natural and immersive user interactions, potentially replacing traditional input methods like keyboards and mice.
- Augmented and Virtual Reality (AR/VR): Hand gesture control will likely become a cornerstone of AR/VR applications, providing users with intuitive ways to manipulate

virtual objects and environments.

2. Healthcare

- Assistive Technologies: Hand sign detection can empower individuals with physical disabilities to interact more effectively with devices and environments, facilitating better communication and control.
- Medical Procedures: Surgeons may use gesture controls for manipulating surgical robots or medical imaging systems, reducing the need for physical contact and enhancing precision.

3. Automotive Industry

- Gesture-Controlled In-Car Systems: Drivers can use gestures to control navigation, entertainment, and climate systems, reducing distractions and enhancing safety.
- Autonomous Vehicles: Gesture recognition can facilitate communication between autonomous vehicles and pedestrians, cyclists, or other drivers, enhancing safety and coordination.

4. Smart Home and IOT

- Home Automation: Hand gestures can control various smart home devices, such as lighting, thermostats, and security systems, offering a more seamless and interactive home environment.
- Elderly and Disability Support: Gesture-based systems can assist the elderly or those with disabilities in managing their home environment more independently.

5. Gaming and Entertainment

- Interactive Gaming: Future gaming systems could utilize hand gesture recognition for more immersive and interactive gameplay experiences.
- Media Control: Users can manage media playback, such as adjusting volume or changing channels, through simple hand gestures, making the experience more user-friendly.

6. Retail and Marketing

- Interactive Displays: Retail environments may feature interactive displays that respond to hand gestures, enhancing customer engagement and experience.

- Personalized Shopping: Hand gesture controls can allow for a more personalized and engaging shopping experience, potentially increasing customer satisfaction and sales.

7. Security and Surveillance

- Enhanced Authentication: Hand gesture recognition can provide an additional layer of security for authentication systems, complementing passwords and biometrics.
- Surveillance Enhancement: Security systems could use gesture recognition to detect unusual or suspicious behavior, improving overall safety and responsiveness.

8. Industrial Automation

- Machinery Control: Workers can control machinery and robots using hand gestures, improving safety and efficiency in industrial environments.
- Maintenance and Training: Gesture recognition can aid in training and maintenance operations, providing hands-free instructions and controls.

9. Education

- Interactive Learning: Hand gesture control can make educational software more interactive and engaging, enhancing the learning experience.
- Accessibility: These systems can make educational tools more accessible to students with disabilities, promoting inclusivity.

10. Public Spaces

- Touchless Interfaces: In public areas like airports, museums, and hospitals, gesture-based interfaces can reduce the need for physical contact with shared surfaces, improving hygiene and user experience.

11. Robotics

- **Human-Robot Interaction:** Robots can be controlled and directed through hand gestures, enhancing the efficiency and intuitiveness of human-robot collaboration.
- **Service Robots:** Gesture control can improve the functionality and user-friendliness of service robots in sectors such as hospitality, healthcare, and customer service.

12. Custom Applications and Innovations

- **Creative Industries:** Artists and designers can use hand gestures to create digital art, manipulate 3D models, and interact with creative software in innovative ways.
- **Personalized Experiences:** Hand sign detection can be tailored to create personalized user experiences across various devices and platforms, adapting to individual user preferences and behaviors.
- Overall, hand sign detection control systems hold the promise of making technology more accessible, intuitive, and efficient, transforming the way we interact with digital and physical worlds. As technology continues to advance, the integration of these systems into everyday life is expected to expand, bringing about significant improvements in convenience, safety, and user experience.

6.2 Applications

Hand sign detection control systems have a wide range of applications across various industries. Here are some notable examples:

1. Human-Computer Interaction

- **Touchless Interfaces:** Enabling users to interact with computers and other devices without physical contact, enhancing convenience and hygiene.
- **Accessibility:** Assisting individuals with disabilities by providing alternative methods for input and control, such as controlling a mouse cursor or typing using hand gestures.

2. Virtual and Augmented Reality (VR/AR)

- **Immersive Experiences:** Allowing users to interact with virtual objects and environments naturally and intuitively through hand gestures.
- **Gaming:** Enhancing gameplay by enabling players to control game actions and navigate menus with hand movements.

3. Healthcare

- **Assistive Technologies:** Helping patients with motor impairments to communicate and interact with devices using hand gestures.

- **Surgical Applications:** Allowing surgeons to manipulate medical images and control surgical tools in a sterile environment without physical contact.

4. Automotive Industry

- **In-Vehicle Controls:** Enabling drivers to control infotainment systems, navigation, and climate settings using hand gestures, reducing distraction and improving safety.
- **Gesture-Based Safety Features:** Allowing for intuitive control of advanced driver-assistance systems (ADAS).

5. Smart Home and IoT

- **Home Automation:** Controlling lights, thermostats, and other smart home devices using hand gestures, offering a more seamless and intuitive user experience.
- **Security Systems:** Integrating gesture control for disarming alarms or unlocking doors.

6. Retail and Marketing

- **Interactive Displays:** Providing customers with interactive product information displays that respond to hand gestures, enhancing engagement.
- **Virtual Try-Ons:** Enabling customers to try on products such as clothes or accessories virtually using gesture-based controls.

7. Education

- **Interactive Learning:** Facilitating interactive teaching tools that respond to hand gestures, making learning more engaging and participatory.
- **Remote Learning:** Enhancing virtual classrooms by allowing teachers and students to interact with content and each other through gestures.

8. Gaming and Entertainment

- **Gesture-Based Controls:** Allowing gamers to control games and consoles using hand movements, offering a more immersive experience.
- **Media Control:** Enabling users to control media playback, volume, and other functions through gestures.

9. Industrial Automation

- **Machinery Control:** Allowing workers to operate machines and equipment using hand gestures, which can improve safety and efficiency.
- **Training:** Providing hands-free operation for training simulations and instructional systems.

10. Public Spaces

- **Information Kiosks:** Offering touchless interaction with public information kiosks in places like airports, museums, and shopping malls.
- **Hygiene Improvement:** Reducing the need for physical contact with surfaces in public areas, which can help in maintaining hygiene.

11. Security and Surveillance

- **Gesture-Based Authentication:** Using hand gestures as a secure method for authentication, adding an additional layer of security.
- **Surveillance Systems:** Detecting suspicious activities through gesture recognition in security cameras.

12. Robotics

- **Human-Robot Interaction:** Enhancing control and communication with robots in industrial, medical, and service settings through intuitive gesture-based commands.
- **Service Robots:** Allowing service robots to interact more naturally with humans in environments like hospitals, hotels, and restaurants.

13. Custom Applications and Innovations

- **Creative Industries:** Facilitating the creation of digital art, 3D modeling, and other creative tasks through gesture control.
- **Personalized User Experiences:** Adapting systems to recognize and respond to individual user gestures, providing customized interactions.

These applications highlight the versatility and potential of hand sign detection control systems to enhance user experience, improve safety, and drive innovation across multiple domains. As technology advances, the integration of gesture recognition into everyday devices and systems is expected to grow, further expanding its applications and impact.

SOURCE CODE

7.1. Hand Detection

```
Import cv2

From cvzone.HandTrackingModule import Hand Detector

Import numpy as np

Offset = 20

ImgSize = 300

cap = cv2.VideoCapture(0)

Detector = HandDetector (maxHands=2)

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)*255

        imgCrop = img[y-offset:y+h+offset, x-offset:x + w+offset]

        imgCropShape = imgCrop.shape  # for height and width

        # imgWhite[0:imgCropShape[0], 0:imgCropShape[1]] = imgCrop

        # Resize imgCrop to match the shape of imgWhite

        imgCropResized = cv2.resize(imgCrop, (imgWhite.shape[1], imgWhite.shape[0])) #
assigning the dimensions values

        # Assign resized imgCrop to imgWhite

        imgWhite[0:imgCropResized.shape[0], 0:imgCropResized.shape[1]] =
imgCropResized
        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("Image White", imgWhite)

    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

7.2. Sign Detection

```
import cv2  # this module is used for capturing image from the cam
from cvzone. HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np   # this module is used for solving matrix problem in programs
import math
# import main
import time

cap = cv2.VideoCapture(0)  # 0 is the id number of webcam of the system
detector = HandDetector(maxHands=1)
classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")

folder = "Data/C"
counter = 0
```

```
labels = ["A", "B", "C"]


offset = 20  # this value will be used for shifting the image size of cropped images
imgSize = 300
while True:
    success, img = cap.read()
    hands, img = detector.findHands(img)
    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']  # bbox means bounding box
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8)*255
        imgCrop = img[y-offset:y + h+offset, x-offset:x + w+offset]   # mentioning the
dimension of image
        imageCropShape = imgCrop.shape


        aspectRatio = h/w  # ratio of height to the width
        # this condition will be used for fixing the height of image
        if aspectRatio > 1:
            k = imgSize/h
            wCal = math.ceil(k*w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            imageResShape = imgResize.shape
            wGap = math.ceil((imgSize-wCal)/2)
            imgWhite[0:imageResShape[0], wGap:wCal+wGap] = imgResize
            prediction, index = classifier.getPrediction(img)
            print(prediction, index)
        # this condition will be used for fixing the width of the image
        else:
            k = imgSize/w
            hCal = math.ceil(k*h)
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))
            imageResShape = imgResize.shape
            hGap = math.ceil((imgSize-hCal)/2)
            imgWhite[hGap:hCal+hGap, :] = imgResize
            prediction, index = classifier.getPrediction(img)
            print(prediction, index)

        cv2.imshow("ImageCrop", imgCrop)  # this function show the crop size of original image
        cv2.imshow("ImageWhite", imgWhite)
    cv2.imshow("Image", img)  # this function shows the original image
    cv2.waitKey(1)
```

7.3. Virtual Volume Increase AND Volume Decrease

```
def volumesigndetecter():
    import cv2
    import mediapipe as mp
    import pyautogui
```

```
x1 = y1 = 0
x2 = y2 = 0
webcam = cv2.VideoCapture(0)
my_hands = mp.solutions.hands.Hands()
drawing_utils = mp.solutions.drawing_utils
while True:
    _, image = webcam.read()
    image = cv2.flip(image, 1)
    frame_height, frame_width, _ = image.shape
    rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    output = my_hands.process(rgb_image)
    hands = output.multi_hand_landmarks
    if hands:
        for hand in hands:
            drawing_utils.draw_landmarks(image, hand)
            landmarks = hand.landmark
            for id , landmark in enumerate(landmarks):  # collecting landmarks from the hands
                x = int(landmark.x * frame_width)
                y = int(landmark.y * frame_height)
                if id == 8:  # for the fourth finger id = 8 means ring finger
                    cv2.circle(img=image, center=(x, y), radius=8, color=(0, 255, 255), thickness=5)
                    x1 = x
```

```
                        y1 = y
                    if id == 4:  # for the thumb finger id = 8 means ring finger
                        cv2.circle(img=image, center=(x, y), radius=8, color=(0, 0, 255), thickness=5)
                        x2 = x
                        y2 = y
            dist = ((x2-x1)**2 + (y2-y1)**2)**(0.5)//4

            cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 5)
            if dist > 50:
                pyautogui.press("volumeup")
            else:
                pyautogui.press("volumedown")

        cv2.imshow("Hand volume using python", image)
        key = cv2.waitKey(10)
        if key == 27:
            break
    webcam.release()
    cv2.destroyAllWindows()

volumesigndetecter()
```

7.4. Virtual Mouse

```
def vir_mouse():
    import cv2
    import mediapipe as mp
    import pyautogui
    cap = cv2.VideoCapture(0)
    hand_detector = mp.solutions.hands.Hands()
    drawing_utils = mp.solutions.drawing_utils
    screen_width, screen_height = pyautogui.size()
    index_y = 0
    while True:
        _, frame = cap.read()
        frame_height, frame_width, _ = frame.shape
        frame = cv2.flip(frame, 1)
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        output = hand_detector.process(rgb_frame)
        hands = output.multi_hand_landmarks
        if hands:
            for hand in hands:
                drawing_utils.draw_landmarks(frame,hand)
                landmarks = hand.landmark
                for id, landmark in enumerate(landmarks):
                    x = int(landmark.x*frame_width)
                    y = int(landmark.y*frame_height)
                # print(x, y)
                if id == 9:
                    cv2.circle(img=frame, center=(x, y), radius=10, color=(0, 255, 255))
                    index_x = screen_width/frame_width*x
                    index_y = screen_height/frame_height*y
                    move To(index_x, index_y)
                if id == 4:
                    cv2.circle(img=frame, center=(x, y), radius=10, color=(0, 255, 255))
                    middle_x = screen_width/frame_width*x
                    middle_y = screen_height/frame_height*y
                    # pyautogui.moveTo(middle_x, middle_y)
                    Print('outside', abs(middle_x - middle_y))
                    if abs(index_y - middle_y) < 25:
                        pyautogui.click(button='left')

        print(hands)
        cv2.imshow('virtual mouse', frame)
        cv2.waitKey(1)


if __name__ == '__main__':
    vir_mouse()
```

7.5. Virtual Keyboard

```
import cv2 as cv

import numpy as np

import imutils

import json

import pyautogui

import time


cam = cv.VideoCapture(0)

arr = []

nums = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "0"]

row1 = ["Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P"]

row2 = ["A", "S", "D", "F", "G", "H", "J", "K", "L"]

row3 = ["Z", "X", "C", "V", "B", "N", "M"]

row4 = ["space", "enter", "backspace", "shift"]

row5 = ["left", "up", "down", "right"]

row6 = ["volumeup", "volumedown", "volumemute"]

x = 10

y = 20

for i in range(0, 10):

    data = {}

    data["x"] = x

    data["y"] = y
    data["w"] = 100

    data["h"] = 80

    data["value"] = nums[i]

    arr.append(data)

    x = x + 100

y = 100

x = 10

for i in range(0, 10):

    data = {}

    data["x"] = x

    data["y"] = y

    data["w"] = 100

    data["h"] = 80

    data["value"] = row1[i]

    arr.append(data)

    x = x + 100

x = 110

y = 180

for i in range(0, 9):

    data = {}

    data["x"] = x

    data["y"] = y

    data["w"] = 100

    data["h"] = 80
    data["value"] = row2[i]
    arr.append(data)
    x = x + 100
x = 210
y = 260
for i in range(0, 7):
    data = {}
    data["x"] = x
    data["y"] = y
    data["w"] = 100
    data["h"] = 80
    data["value"] = row3[i]
    arr.append(data)
    x = x + 100
x = 110
y = 340
data = {}
data["x"] = x
data["y"] = y
data["w"] = 200
data["h"] = 80
data["value"] = row4[0]
arr.append(data)
data = {}
```

```
data["x"] = 310

data["y"] = y

data["w"] = 200

data["h"] = 80

data["value"] = row4[1]

arr.append(data)

data = {}

data["x"] = 510

data["y"] = 340

data["w"] = 250

data["h"] = 80

data["value"] = row4[2]

arr.append(data)

data = {}

data["x"] = 760

data["y"] = 340

data["w"] = 200

data["h"] = 80

data["value"] = row4[3]

arr.append(data)

x = 110

y = 420

data = {}

data["x"] = x
```

```
data["y"] = y

data["w"] = 200

data["h"] = 80

data["value"] = row5[0]

arr.append(data)

data = {}

data["x"] = 310

data["y"] = y

data["w"] = 200

data["h"] = 80

data["value"] = row5[1]

arr.append(data)

data = {}

data["x"] = 510

data["y"] = y

data["w"] = 200

data["h"] = 80

data["value"] = row5[2]

arr.append(data)

data = {}

data["x"] = 710

data["y"] = y

data["w"] = 200

data["h"] = 80
```

```
data["value"] = row5[3]

arr.append(data)

x = 10

y = 500

data = {}

data["x"] = x

data["y"] = y

data["w"] = 200

data["h"] = 80

data["value"] = row6[0]

arr.append(data)

data = {}

data["x"] = 210

data["y"] = y

data["w"] = 200

data["h"] = 80

data["value"] = row6[1]

arr.append(data)

data = {}

data["x"] = 410

data["y"] = y

data["w"] = 200

data["h"] = 80

data["value"] = row6[2]
```

```
arr.append(data)


json_string = json.dumps(arr)

json_data = json.loads(json_string)

# print(json_data)

while (1):

    ret, img = cam.read()

    img = cv.GaussianBlur(img, (5, 5), 0)

    img = imutils.resize(img, width=1030, height=700)

    height, width = img.shape[:2]

    x = 10

    y = 20

    for i in range(0, 10):

        cv.rectangle(img, (x, y), (x + 100, y + 80), (0, 255, 255), 2)

        cv.putText(img, nums[i], (x + 50, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0,
255), 2, cv.LINE_AA, False)

        x = x + 100

    y = 100

    x = 10

    for i in range(0, 10):

        cv.rectangle(img, (x, y), (x + 100, y + 80), (0, 255, 255), 2)

        cv.putText(img, row1[i], (x + 50, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0,
255), 2, cv.LINE_AA, False)

        x = x + 100
    x = 110
    y = 180
    for i in range(0, 9):
        cv.rectangle(img, (x, y), (x + 100, y + 80), (0, 255, 255), 2)
        cv.putText(img, row2[i], (x + 50, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
        x = x + 100

    x = 210
    y = 260
    for i in range(0, 7):
        cv.rectangle(img, (x, y), (x + 100, y + 80), (0, 255, 255), 2)
        cv.putText(img, row3[i], (x + 50, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
        x = x + 100
    x = 110
    y = 340
    cv.rectangle(img, (x, y), (x + 200, y + 80), (0, 255, 255), 2)
    cv.putText(img, row4[0], (x + 70, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
    x = 310
    y = 340
    cv.rectangle(img, (x, y), (x + 200, y + 80), (0, 255, 255), 2)
    cv.putText(img, row4[1], (x + 70, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
    x = 510
    y = 340
    cv.rectangle(img, (x, y), (x + 250, y + 80), (0, 255, 255), 2)
    cv.putText(img, row4[2], (x + 70, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
    x = 760
    y = 340
    cv.rectangle(img, (x, y), (x + 200, y + 80), (0, 255, 255), 2)
    cv.putText(img, row4[3], (x + 70, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
    x = 110
    y = 420
    cv.rectangle(img, (x, y), (x + 200, y + 80), (0, 255, 255), 2)
    cv.putText(img, row5[0], (x + 100, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
    x = 310
    y = 420
    cv.rectangle(img, (x, y), (x + 200, y + 80), (0, 255, 255), 2)
    cv.putText(img, row5[1], (x + 100, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
    x = 510
    y = 420
    cv.rectangle(img, (x, y), (x + 200, y + 80), (0, 255, 255), 2)
    cv.putText(img, row5[2], (x + 100, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
    x = 710
    y = 420
    cv.rectangle(img, (x, y), (x + 200, y + 80), (0, 255, 255), 2)
    cv.putText(img, row5[3], (x + 100, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
    x = 10
    y = 500
    cv.rectangle(img, (x, y), (x + 200, y + 80), (0, 255, 255), 2)
    cv.putText(img, "V+", (x + 60, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
    x = 210
    y = 500
    cv.rectangle(img, (x, y), (x + 200, y + 80), (0, 255, 255), 2)
    cv.putText(img, "V-", (x + 60, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)
    x = 410
    y = 500
    cv.rectangle(img, (x, y), (x + 200, y + 80), (0, 255, 255), 2)
    cv.putText(img, "Vmute", (x + 60, y + 40), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE_AA, False)

    hsv_img = cv.cvtColor(img, cv.COLOR_BGR2HSV)
    mask = cv.inRange(hsv_img, np.array([65, 60, 60]), np.array([80, 255, 255]))
    mask_open = cv.morphologyEx(mask, cv.MORPH_OPEN, np.ones((5, 5)))
    mask_close = cv.morphologyEx(mask_open, cv.MORPH_CLOSE, np.ones((20, 20)))
    mask_final = mask_close
    conts, _ = cv.findContours(mask_final.copy(), cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
    cv.drawContours(img,conts,-1,(255,0,0),3)
    if (len(conts) == 1):
        x, y, w, h = cv.boundingRect(conts[0])
        cx = round(x + w / 2)
        cy = round(y + h / 2)
        cv.circle(img, (cx, cy), 20, (0, 0, 255), 2)
        word = ""
        for i in range(len(json_data)):
            if cx >= int(json_data[i]["x"]) and cx <= int(json_data[i]["x"]) +
int(json_data[i]["w"]) and cy >= int(
                    json_data[i]["y"]) and cy <= int(json_data[i]["y"]) + int(json_data[i]["h"]):
                word = json_data[i]["value"]
        # print(word)
        pyautogui.press(word)
        # pyautogui.PAUSE=2.5
        # time.sleep(1)


    cv.imshow("cam", img)
    cv.waitKey(10)
```

7.6. Virtual Scroll UP and Scroll Down

```
import pyautogui

import virtualmouse




def model_():

    import cv2  # this module is used for capturing image from the cam

    from cvzone .Hand Tracking  Module import HandDetector

    from cvzone.ClassificationModule import Classifier

    import numpy as np   # this module is used for solving matrix problem in programs

    import math

    import time


    cap = cv2.VideoCapture(0) # 0 is the id number of webcam of the system

    detector = HandDetector(maxHands=1)

    classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")


    folder = "Data/C"

    counter = 0

    labels = ["A", "B", "C"]


    offset = 20 # this value will be used for shifting the image size of cropped images

    imgSize = 300

    while True:
```

```
    success, img = cap.read()

    hands, img = detector.findHands(img)

    if hands:

        hand = hands[0]

        x, y, w, h = hand['bbox'] # bbox means bounding box

        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8)*255

        imgCrop = img [y-offset:y + h+offset, x-offset:x + w+offset] # mentioning the
dimension of image

        imageCropShape = imgCrop.shape


        aspectRatio = h/w # ratio of height to the width

        # this condition will be used for fixing the height of image

        if aspectRatio > 1:

            k = imgSize/h

            wCal = math.ceil(k*w)

             imgResize = cv2.resize(imgCrop, (wCal, imgSize))

            imageResShape = imgResize.shape

            wGap = math.ceil((imgSize-wCal)/2)

            imgWhite[0:imageResShape[0], wGap:wCal+wGap] = imgResize

            prediction, index = classifier.getPrediction(img)

            print(prediction, index)

            if index == 0:

                pyautogui.scroll(-100)

            elif index == 1:
```

```
            pyautogui.scroll(100)
        elif index == 2:
            break
    # this condition will be used for fixing the width of the image
    else:
        k = imgSize/w
        hCal = math.ceil(k*h)
        imgResize = cv2.resize(imgCrop, (imgSize, hCal))
        imageResShape = imgResize.shape
        hGap = math.ceil((imgSize-hCal)/2)
        imgWhite[hGap:hCal+hGap, :] = imgResize
        prediction, index = classifier.getPrediction(img)
        print(prediction, index)
        if index == 0:
            pyautogui. scroll(-100)
        elif index == 1:
            pyautogui scroll(100)
        elif index == 2:
            break

    cv2.imshow("ImageCrop", img Crop) # this function show the crop size of original image
    cv2.imshow("ImageWhite", imgWhite)
cv2.imshow("Image", img) # this function shows the original image
```

```
        cv2.waitKey(1)


if __name__ == '__main__':

    model ()
```

SOURCE CODE OUTPUT

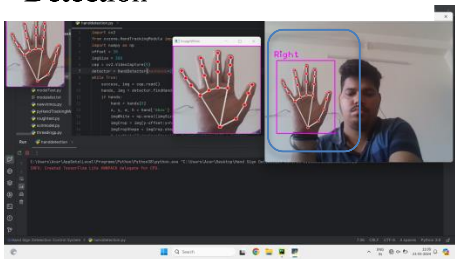


Figure 7.1:- Right Hand Detection

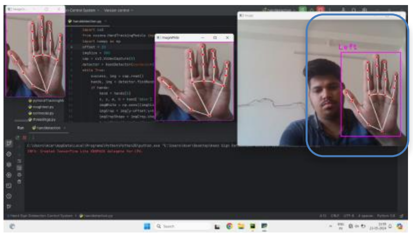


Figure 7.2:- left Hand Detection

Sign Detection

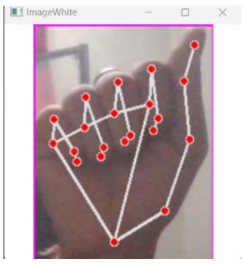


Figure 7.1:- Sign Detection of A

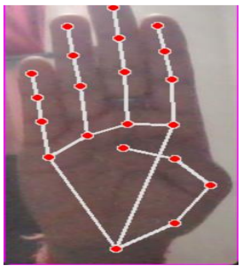


Figure 7.1:- Sign Detection of B

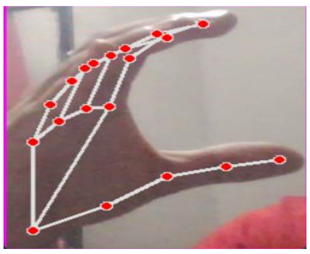


Figure 7.1:- sign Detection of C

Virtual Volume Increase and Volume Decrease

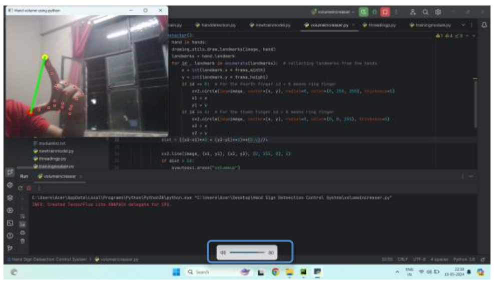


Figure 7.1:- Virtual Volume Increased

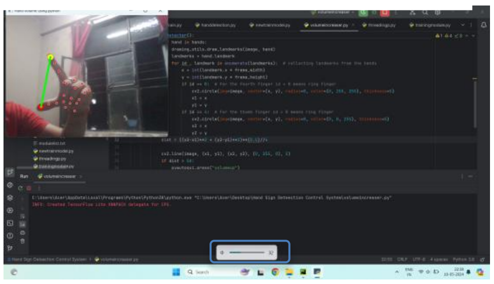
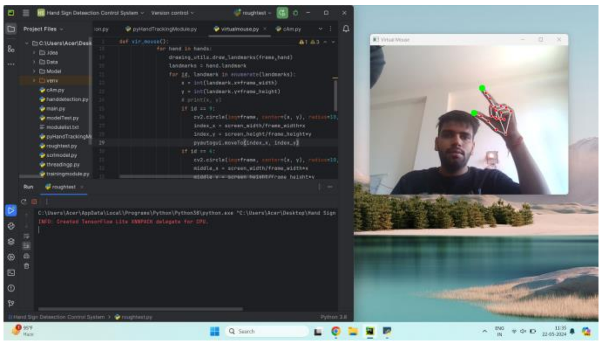


Figure:-7.1 Virtual Volumes Decreased

Virtual Mouse



```
def vir_mouse():
            for hand in hands:
                drawing_utils.draw_landmarks(frame,hand)
                landmarks = hand.landmark
                for id, landmark in enumerate(landmarks):
                    x = int(landmark.x*frame_width)
                    y = int(landmark.y*frame_height)
                    # print(x, y)
                    if id == 8:
                        cv2.circle(img=frame, center=(x, y), radius=10,
                        index_x = screen_width/frame_width*x
                        index_y = screen_height/frame_height*y
                        pyautogui.moveTo(index_x, index_y)
                    if id == 4:
                        cv2.circle(img=frame, center=(x, y), radius=10,
                        middle_x = screen_width/frame_width*x
                        middle_y = screen_height/frame_height*y
```

```
C:\Users\Acer\AppData\Local\Programs\Python\Python38\python.exe "C:\Users\Acer\Desktop\Hand Sign
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
```

Figure 7.1:- Virtual Mouse

Virtual Keyboard

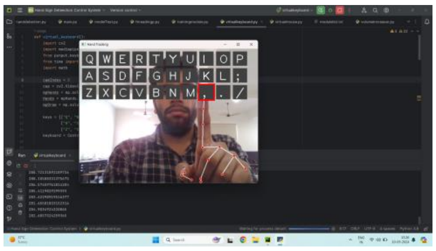
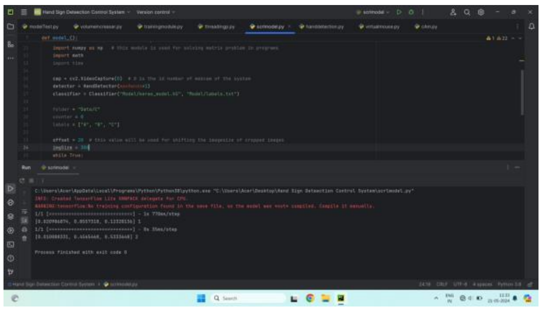


Figure 7.1:- Virtual Keyboard

Virtual Scroll UP and Scroll Down



```
def model_():
    import numpy as np  # This module is used for solving matrix problem in program
    import math
    import time

    cap = cv2.VideoCapture(0)  # 0 is the id number of webcam of the system
    detector = HandDetector(maxHands=1)
    classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")

    folder = "Data/C"
    counter = 0
    labels = ["A", "B", "C"]

    offset = 20  # this value will be used for shifting the imagesize of cropped images
    imgSize = 300
    while True:
```

```
C:\Users\Acer\AppData\Local\Programs\Python\Python38\python.exe "C:\Users\Acer\Desktop\Hand Sign Detection Control System\scrlmodel.py"
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
1/1 [==============================] - 1s 770ms/step
[0.020986874, 0.8557318, 0.12328136] 1
1/1 [==============================] - 0s 35ms/step
[0.010088331, 0.4565668, 0.5333668] 2

Process finished with exit code 0
```

Figure 7.1:- Virtual Scroll UP and Scroll Down

REFRENCE

[1] G. R. S. Murthy, R. S. Jadon. (2009). "A Review of Vision Based Hand Gestures Recognition," International Journal of Information Technology and Knowledge Management, vol. 2(2), pp. 405-410.

[2] P. Garg, N. Aggarwal and S. Sofat. (2009). "Vision Based Hand Gesture Recognition," World Academy of Science, Engineering and Technology, Vol. 49, pp. 972-977.

[3] Fakhreddine Karray, Milad Alemzadeh, Jamil Abou Saleh, Mo Nours Arab, (2008) "Human-Computer Interaction: Overview on State of the Art", International Journal on Smart Sensing andIntelligent Systems, Vol. 1(1)

[4] Joseph J. LaViola Jr., (1999). "A Survey of Hand Posture and Gesture Recognition Techniques andTechnology", Master Thesis, Science and Technology Center for Computer Graphics and Scientific Visualization, USA.

[5] Rafiqul Z. Khan, Noor A. Ibraheem, (2012). "Survey on Gesture Recognition for Hand Image Postures", International Journal of Computer And Information Science, Vol. 5(3), Doi: 10.5539/cis.v5n3p110

[6] Thomas B. Moeslund and Erik Granum, (2001). "A Survey of Computer Vision-Based Human Motion Capture," Elsevier, Computer Vision and Image Understanding, Vol. 81, pp.
231–268.

[7] Freeman, W. T., Weissman, C. D. (1995). " Television Control by Hand Gestures". IEEE
International Workshop on Automatic Face and Gesture Recognition.

[8] V. S. Kulkarni, S.D.Lokhande, (2010) "Appearance Based Recognition of American Sign Language

Using Gesture Segmentation”, International Journal on Computer Science and Engineering (IJCSE),
Vol. 2(3), pp. 560-565.

[9] Malima, A., Özgür, E., Çetin, M. (2006). “A Fast Algorithm for Vision-Based Hand Gesture Recognition For Robot Control”, IEEE 14th conference on Signal Processing and Communications Applications, pp. 1-4. doi: 10.1109/SIU.2006.1659822

[10] Mokhar M. Hasan, Pramod K. Mishra, (2012). “Robust Gesture Recognition Using Gaussian
Distribution for Features Fitting’, International Journal of Machine Learning and Computing, Vol.
2(3).

[11] W. T. Freeman and Michal R., (1995) “Orientation Histograms for Hand Gesture Recognition”,
IEEE International Workshop on Automatic Face and Gesture Recognition.

[12] Min B., Yoon, H., Soh, J., Yangc, Y., & Ejima, T. (1997). “Hand Gesture Recognition Using Hidden Markov Models”. IEEE International Conference on computational cybernetics and simulation. Vol. 5, Doi: 10.1109/ICSMC.1997.637364
[13] Verma, R., Dev A. (2009).”Vision based hand gesture recognition using finite state machines and fuzzy logic”. IEEE International Conference on Ultra-Modern Telecommunications & Workshops (ICUMT '09), pp. 1-6. doi: 10.1109/ICUMT.2009.5345425

[14] Luigi Lamberti, Francesco Camastra, (2011). “Real-Time Hand Gesture Recognition Using a Color Glove”, Springer Proceedings of the 16th international conference on Image analysis and processing: Part I ICIAP.

[15] Minghai Y., Xinyu Q., Qinlong G., Taotao R., Zhongwang L., (2010). “Online PCA with Adaptive Subspace Method for Real-Time Hand Gesture Learning and Recognition”, journal World Scientific and Engineering Academy and SocietWSEAN, Vol. 9(6).

[16] N. A. Ibraheem., R. Z. Khan, (2012). “Vision Based Gesture Recognition Using Neural Networks Approaches: A Review”, International Journal of Human Computer Interaction (IJHCI), Malaysia, Vol. 3(1).

[17] Cheng-Chang L. and Chung-Lin H., (1999).“The Model-Based Dynamic Hand Posture Identification
Using Genetic Algorithm”, Springer, Machine Vision and Applications Vol. 11.

[18] Kouichi M., Hitomi T. (1999) “Gesture Recognition using Recurrent Neural Networks” ACM
conference on Human factors in computing systems: Reaching through technology (CHI '91), pp.
237-242. doi: 10.1145/108844.108900

[19] Guan, Y., Zheng, .M. (2008). “Real-time 3D pointing gesture recognition for natural HCI. IEEE
Proceedings of the 7th World Congress on Intelligent Control and Automation WCICA 2008, doi:
10.1109/WCICA.2008.4593304

[20] Freeman, W. T., Weissman, C. D. (1995). ” Television Control by Hand Gestures”. IEEE
International Workshop on Automatic Face and Gesture Recognition
Gurjal, P. and Kunnur, K. Real time Hand Gesture Recognition using SIFT. Int. J. of Electronics & Electrical Engg. 2(3) (March 2012), 19 – 33

[21] Rautaray, S. S. and Agarwal, A. Real time Hand Gesture Recognition System for Dynamic Applications. Int. J. of UbiComp, 3(1) (January 2012), 21 – 31

[22] Sultana, A. and Rajapuspha, T. Vision based Gesture Recognition for Alphabetical Hand Gestures using the SVM Classifier. Int. J. of Comp. Sc. And Engg. Tech. 3(7) (July 2012), 218 – 223

[23] Ghotkar, A. S. and Kharate, G. K. Hand Segmentation Techniques to Hand Gesture Recognition for Natural Human Computer Interaction. Int. J. of Human Computer Interaction, 3(1) (2012) 15 - 25

[24] Kristensson, P. O., Nicholson, T. F. W. and Quigley, A. 2012. Continuous Recognition of One-handed and Twohanded Gestures using 3-D Full-body motion tracking sensors. In Proc. of IUI 12

[25] Rautaray, S. S. Real time Multiple Hand Gesture Recognition System for Human Computer Interaction. Int. J. of Intelligent Systems and Applications, 5 (May 2012), 56 - 64

[26] Kishore, P. V. V. and Kumar, P. R. Segment, Track, Extract, Recognize and Convert Sign Language Videos to Voice/ Text. Int. J. of Advance Comp. Sc. and Applications, 3(6) (2012) 35 - 47

[27] Trindade, P., Lobo, J. and BaHrreto, J. P. 2012. Hand Gesture Recognition using Color and Depth Images Enhanced with Hand Angular Pose Data. In Proc. of IEEE Int. Conf. on Multisensor Fusion & Integration for Intelligent Systems

[28] Singh, S., Jain, A. and Kumar, D. Recognizing and Interpreting Sign Language Gesture for Human Robot Interaction. Int. J. of Computer Applications, 52 (11) (August 2012) 24 - 30

[29] Lahamy, H. and Lichti, D. 2012. Robust Real-Time and Rotation Invariant American Sign Language Alphabet Recognition using Range Camera. In Proc. of XXII ISPRS Congress

[30] Kishore, P. V. V. and Kumar, P. R. A Model for Real Time Sign Lang Recognition System. Int. J. of Adv. Research in Comp. Sc. And Software Engg. 2 (6) (June 2012), 29 – 35

[31] Sharma, N. and Sharma, H. HIM: Hand Gesture Recognition in Mobile-learning. Int. J. of Comp. Applications, 44(16) (April 2012), 33 – 37

[32] Ghotkar, A. S., Khatal, R., Khupase, S., Asati, S. and Hadap, M. 2012. Hand Gesture Recognition for Indian Sing Language. In Proc. of Int. Conf. on Computer Communication and Informatics

[33] Bui, T. T. T., Phan, N. H. and Spitsyn, V. G. 2012. Face and Hand Gesture Recognition Algorithm Based on Wavelet transforms and Principal Component Analysis. In Proc. of 7th Int. Conf. on Strategic Technology.

[34] Kohn, B., Belbachir, A. N. and Nowakowska, A. 2012. Real-time Gesture Recognition using bio inspired 3D Vision Sensor. In Proc. of IEEE Comp. Society Conference on Comp. Vision and Pattern Recognition Workshops

[35] Kishore, P. V. V. and Kumar, P. R. A Video Based Indian Sign Language Recognition System (INSLR) Using Wavelet Transform and Fuzzy Logic. Int. J. of Engg. and Tech. 4(5) (October 2012), 537 - 542

[36] Geetha, M. and Manjusha, U. C. A Vision Based Recognition of Indian Sign Language Alphabets and Numerals Using B-Spline Approximation. Int. J. On Comp. Sc. and Engg. 4(3) (March 2012), 406 - 415

[37] Kurakin, A., Zhang, Z. and Liu, Z. 2012. A Real Time System for Dynamic Hand Gesture Recognition with a Depth Sensor. In Proc. Of 20th European Signal Processing Conference.

[38] Khan, R. Z. and Ibraheem, N. A. Hand Gesture Recognition: A Literature Review. Int. J. of Artificial Intelligence and Applications, 3(4) (July 2012), 161-173

[39] Suarez, J. and Murphy, R. R. 2012. Hand Gesture Recognition with Depth Images: A Review. In Proc. of Int. Sym. on Robot and Human Interactive Communication

[40] Kasprzak, W., Wilkowski, A. and Czapnik, K. Hand Gesture Recognition based on Free-Form Contours and Probabilistic Inference. Int. J. of Applied Math. and Comp. Sc. 22(2) (2012), 437-

[41] Jalab, H. A., Static Hand Gesture Recognition for Human Computer Interaction. Information Technology J. 11(09) (2012), 1265 - 1271

[42] Gowtham, P. N. V. S. A Hand Gesture Recognition based Virtual Touch World. Int. J. of Information & Education Technology, 2(1) (Feb 2012), 36 – 4

[43] Shen, X., Hua, G., Williams, L. and Wu, Y. Dynamic Hand Gesture Recognition: An exemplar-based approach from motion divergence fields. J. of Image and Vision Computing, 30(3) (March 2012), 227 – 235

[44] Panwar, M. 2012. Hand Gesture Recognition based on Shape Parameters. In Proc. of Int. Conf. on Computing, Communication and Applications

[45] Pradhan, A., Ghose, M. K. and Pradhan, M. Hand Gesture Recognition using Feature Extraction. Int. J. of Current Engg. And Tech. 2(4) (Dec. 2012), 323 – 327

[46] Caputo, M., Denker, K., Dums, B. and Umlauf, G. 2012. 3D Hand Gesture Recognition based on Sensor Fusion of Commodity Hardware. In Proc. of Mensch & Computer

[47] Dardas, N. H. and Georganas, N. D. Real-time Hand Gesture Detection and Recognition using Bag-ofFeatures and Support Vector Machine Techniques. IEEE Trans. on Instrumentation and Measurement, 60(11) (Nov 2011), 3592 – 3607

[48] Panwar, M. and Mehra, P. S. 2011. Hand Gesture Recognition for Human Computer Interaction. In Proc. of Int. Conf. on Image Information Processing

[49] Ren, Z., Yuan, J and Zhang, Z. 2011. Robust Hand Gesture Recognition Based on Finger-Earth Mover’s Distance with Commodity Depth Camera. In Proc. of the 19th ACM Int. Conf. on Multimedia

[50] Meena, S. 2011. A Study on Hand Gesture Recognition Technique. M. Tech. thesis. NIT, Rourkela (India)

[51] Bhuyan, M. K., Kar, M. K. and Neog, D. R. 2011. Hand Pose Identification from Monocular Image for Sign.

Yoon, H., Soh, J., Bae, Y. J. and Yang, H. S. Hand Gesture Recognition using Combined Features of Location, Angle and Velocity. J. of Pattern Recognition Society, 34 (2001)

[52] Jeong, M. H., Kuno, Y. and Shimada, N. 2001. Recognition of Shape-Changing Hand Gestures Based on Switching Linear Model. 11th Int. Conf. on Image Analysis and Processing

[53] Lamar, M.V., Bhuiyan, Md. S. and Iwata, A. 1999. Hand Gesture Recognition using Morphological Principal Component Analysis and an Improved CombNET-II. In Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (Vol. 4)

[54] Liang, R. and Ouhyoung, M. 1998. A Real-time Continuous Gesture Recognition System for Sign Language. In Proc. 3rd IEEE Int. Conf. on Automatic Face and Gesture Recognition

[55] Min, B., Yoon, H., Soh, J., Yang, Y. and Ejima, T. 1997. Hand Gesture Recognition using Hidden Markov Models. In Proc. of IEEE Int. Conf. on Systems, Man, & Cybernetics (Vol. 5)

[56]] E. Ohn-Bar and M. M. Trivedi. Hand gesture recognition in real time for automotive interfaces: A multimodal visionbased approach and evaluations. IEEE Trans. on Intelligent Transportation Systems, 15(6):2368–2377, 2014.

[57] X. Peng, C. Zou, Y. Qiao, and Q. Peng. Action recognition with stacked fisher vectors. In Computer Vision–ECCV 2014, pages 581–595. Springer, 2014.

[58] F. Perronnin, J. Sanchez, and T. Mensink. Improving the ´ fisher kernel for large-scale image classification. In Computer Vision–ECCV, pages 143–156. Springer, 2010.
[59] N. Pugeault and R. Bowden. Spelling it out: Real-time asl fingerspelling recognition. In IEEE computer Vision Workshops (ICCV Workshops), pages 1114–1119, Nov 2011.

[60] Z. Ren, J. Yuan, and Z. Zhang. Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera. In ACM International Conference on Multimedia, MM '11, pages 1093–1096, New York, NY, USA, 2011. ACM.

[61] J. Sanchez, F. Perronnin, T. Mensink, and J. Verbeek. Im- ´ age classification with the fisher vector: Theory and practice. International journal of computer vision, 105(3):222–245, 2013.

[62] T. Sharp, C. Keskin, D. Robertson, J. Taylor, J. Shotton, D. Kim, C. Rhemann, I. Leichter, A. Vinnikov, Y. Wei, D. Freedman, P. Kohli, E. Krupka, A. Fitzgibbon, and S. Izadi. Accurate, robust, and flexible real-time hand tracking. CHI, April 2015.

[63] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from a single depth image. In Computer Vision on Pattern Recognition (CVPR), June 2011.

[64] R. Slama, H. Wannous, M. Daoudi, and A. Srivastava. Accurate 3d action recognition using learning on the grassmann manifold. Pattern Recognition, 48(2):556 – 567, 2015.

[65] H.-I. Suk, B.-K. Sin, and S.-W. Lee. Hand gesture recognition based on dynamic bayesian network framework. Pattern Recognition, 43(9):3059 – 3072, 2010.

[66] D. Tang, H. J. Chang, A. Tejani, and T. K. Kim. Latent regression forest: Structured estimation of 3d articulated hand posture. In Computer Vision and Pattern Recognition (CVPR), pages 3786–3793, June 2014.

[67] H. Wang, Q. Wang, and X. Chen. Hand posture recognition from disparity cost map. In ACCV (2), volume 7725 of Lecture Notes in Computer Science, pages 722–733. Springer, 2012.

[68] J. Wang, Z. Liu, Y. Wu, and J. Yuan. Mining actionlet ensemble for action recognition with depth cameras. In Computer Vision and Pattern Recognition (CVPR), pages 1290–1297, June 2012.

[69] C. Zhang, X. Yang, and Y. Tian. Histogram of 3d facets: A characteristic descriptor for hand gesture recognition. In IEEE Int. Conference and Workshops on Automatic Face and Gesture Recognition (FG),, pages 1–8, April 2013.