A Project Report
on
VIRTUAL ACCESS USING MACHINE LEARNING

A project report submitted in partial fulfillment of the Academic requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

By

N. Ajay	(17TQ1A0562)
Ch. Ruchitha	(17TQ1A0517)
A. Saikumar	(17TQ1A0506)
K. Abhilash	(17TQ1A0546)
K. Shiva Kumar	(18TQ5A0506)

Under the guidance of

Miss. J. Sabitha, Asst. Prof.

Department Of Computer Science and Engineering

SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES

(Approved by AICTE & Affiliated to JNTUH)

Korremula Road, Narapally, Ghatkesar, R.R District-501301

(2020-2021)

SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES

(Approved by AICTE, Affiliated to JNTU Hyderabad, Accredited by NAAC)

Korremula Road, Narapally (V), Ghatkesar Mandal,Medchal-Dist:- 500088

CERTIFICATE

This is certify that the project report titled **“VIRTUAL ACCESS USING MACHINE LEARNING”** is being submitted by **N. AJAY (17TQ1A0562), CH. RUCHITHA (17TQ1A0517), A. SAI KUMAR (17TQ1A0506), K. ABHILASH, (17TQ1A0546), K. SHIVA KUMAR (18TQ5A0506)**, in B Tech IV II semester **Computer Science & Engineering** is a record bonafide work carried out by them. The results embodied in this report have not been submitted to any other University for the award of any degree.

Internal Guide	**Head of the Department**	**Principal**
Miss. J. SABITHA	Dr. A. SATYANARAYANA	Dr. Y.RAJA SREE RAO
Asst Professor		

External Examiner

PROJECT HELD ON:

DECLARATION

We **N. AJAY, CH. RUCHITHA, A. SAI KUMAR, K. ABHILASH, K. SHIVA KUMAR** hereby declared that the results embodied in this project dissertation entitled "**VIRTUAL ACCESS USING MACHINE LEARNING**" is carried out by us during the year 2020-2021 in partial fulfillment of the award of Bachelor of Technology in **Computer Science and Engineering** from **Siddhartha Institute of Technology and Science**, It is an authentic record carried by us under the guidance of **Miss. SABITHA** Assistant professor, Department of **COMPUTER SCIENCE AND ENGINEERING**.

Date:

Place: Narapally

N. Ajay	**(17TQ1A0562)**
Ch. Ruchitha	**(17TQ1A0517)**
A. Saikumar	**(17TQ1A0506)**
K. Abhilash	**(17TQ1A0546)**
K. Shiva Kumar	**(18TQ5A0506)**

ACKNOWLEDGEMENT

This is an acknowledgement of the intensive drive and technical computer of many individual who have contributed to the success of our project.

We heartily thankful to our principal **Dr. Y. RAJASREE RAO** and Head of the Department **Dr. A. SATYANARAYANA**, Computer Science and Engineering, for there constant support towards our project.

We are grateful to our guide **Miss. J. SABITHA** , Assistant Professor, Computer Science and Engineering Department, who gave us the necessary motivation and support during the full course of our project.

We would like to express our immense gratitude and sincere thanks to all the faculty members of CSE department and our friends for their valuable suggestions and support which directly or indirectly helped us in successful completion of work.

N. Ajay	**(17TQ1A0562)**
Ch. Ruchitha	**(17TQ1A0517)**
A. Saikumar	**(17TQ1A0506)**
K. Abhilash	**(17TQ1A0546)**
K. Shiva Kumar	**(18TQ5A0506)**

Vision of the Department: To be a Recognized Center of Computer Science Education with values and quality research

Mission of the Department:

MISSION	STATEMENT
DM1	Impart High Quality Professional Training With An Emphasis On Basic principles Of Computer Science And Allied Engineering
DM2	Imbibe Social Awareness And Responsibility To Serve The Society
DM3	Provide Academic Facilities Organize Collaborated Activities To enable Overall Development Of Stakeholders

Programme Educational Objectives (PEOs)

- **PEO1:**Graduates will be able to synthesize mathematics, science, engineering fundamentals, laboratory and work- based experiences to formulate and to solve problems proficiently in Computer science and Engineering and related domains.
- **PEO2:** Graduates will be prepared to communicate effectively and work in multidisciplinary engineering projects following the ethics in their profession.
- **PEO3:** Graduates will recognize the importance of and acquire the skill of independent learning to shine as experts in the field with a sound knowledge.

ABSTRACT

In this modern age the advancement in ubiquitous computing has made the use of natural user interface very much required. The `ad used and are still using a broad range of gestures to communicate or interact with each other. Human gesture is a mode of non - verbal interaction medium and can provide the most intuitive, originative and natural way to interact with computers. Our main goal is to make the interaction between human and computer as natural as the interaction between humans. The objective of this paper is to recognize the static hand gesture images (i.e. frames) based on shapes and orientations of hand which is extracted from input video stream recorded in stable lighting and simple background conditions. We can use this vision based recognized gestures to control multimedia applications (like Windows Media Player, Windows Picture Manager, VLC Player etc.) running on computer using different gestural commands.

CONTENTS

LIST OF FIGURES

LIST OF SCREENS

Chapter 1
INTRODUCTION

1.1 Motivation:

With the massive influx and advancement of technologies, a computer system has become a very powerful machine which has been designed to make the human beings' tasks easier. Due to which the HCI (human – computer interaction) has become an important part of our lives. Now-a-days, the progress and development in interaction with computing devices has increased so fast that as a human being even we could not remained left with the effect of this and it has become our primary thing. The technologies has so much surrounded us and has made a place in our lives that we use it to communicate, shop, work and even entertain ourselves1. There are many applications like media player, MS-office, Windows picture manager etc. which require natural and intuitive interface. Now-a-days most of the users uses keyboard, mouse, pen, Joysticks etc. to interact with computers, which are not enough for them. In the near future, these existing technologies which are available for the computing, communication and display will become a bottleneck and the advancement in these technologies will be required to make the system as natural as possible. Nevertheless the invention of mouse and keyboards by the researchers and engineers has been a great progress, there are still some situations where interaction with computer with the help of keyboard and mouse will not be enough.

This project is therefore aimed at investigating and developing a Computer Control (CC) system using hand gestures. Most laptops today are equipped with webcams, which have recently been used insecurity applications utilizing face recognition. In order to harness the full potential of a webcam, it can be used for vision based CC, which would effectively eliminate the need for a computer mouse or mouse pad. The usefulness of a webcam can also be greatly extended to other HCI application such as a sign language database or motion controller. .

1.2 Methodology:

The system we have proposed and designed for vision-based hand gesture recognition system contained various stages which we have explained through an algorithm. The working flow-chart of gesture recognition system has also shown in Fig. 1.

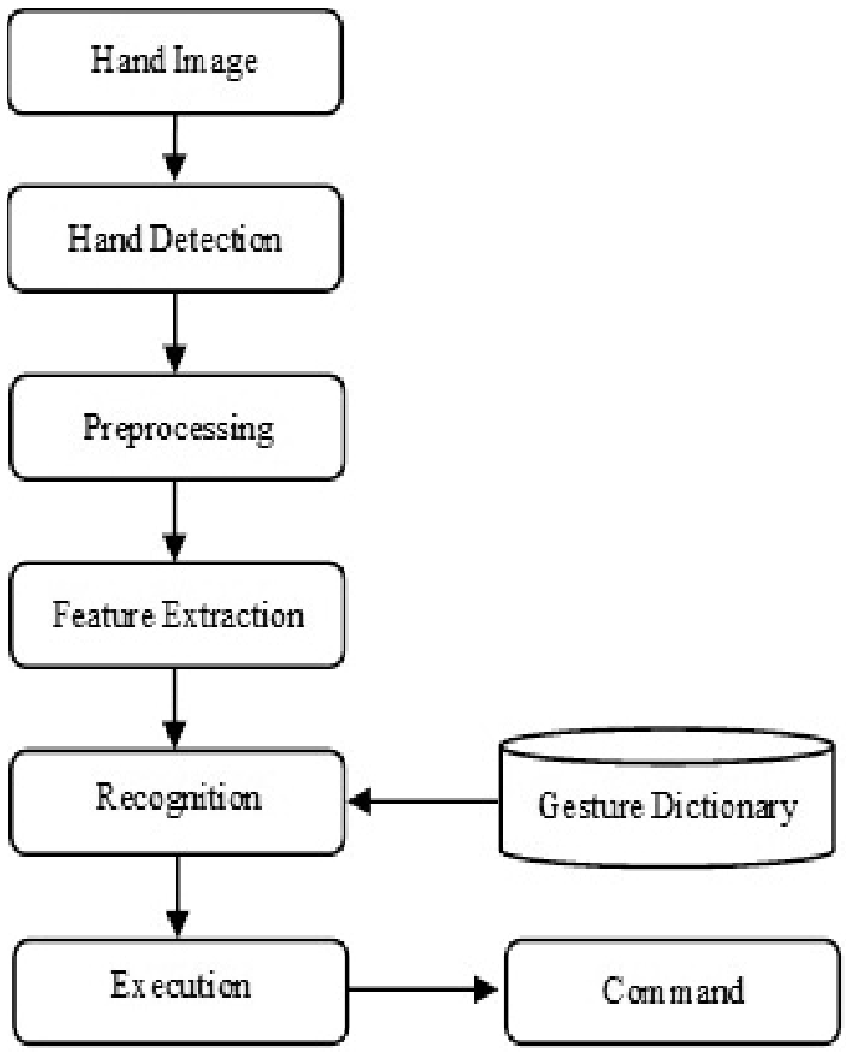


Fig 1.2 Methodology

When the User Inputs the Hand Image the hand detection algorithm will trigger which will further some processing at preprocessing stage like subtraction of background Image and ensure that Input Image do contain some Hand input which later will be given for the recognizing the user gesture.

This Gesture will be carried by matching the gesture configured from the Gesture Dictionary then configured command execution will be carried out as the final output.

1.3 Virtual Access Using Hand Gesture, Facial Expression:

According to **Markets and markets** , the market for gesture recognition is expected to grow from USD 9.6 Billion in 2020 to USD 32.3 Billion in just 5 years. More and more companies are adopting the technology to ease the lives of their customers and solve some of their everyday problems. This is a win-win situation

Gestures play an important role in our everyday communication and expression. Thus using them to communicate with tech devices needs very little intellectual data processing from our side. That means we can control different things such as vending machines almost without thinking, just by using our fingers and hands.

We have developed a prototype of the touchless interface system which uses our hand tracking and gesture recognition technology. The user can interact with a screen without touching it by taking advantage of specific gestures to browse items, select a particular folder and controlling the volume while some video is playing the media player.

This prototype can be further improved and enhanced in order to use this in the many real time applications.

1.4 Need for Hand Gesture Recognition Technology:

Many of us were happy to stop using physical buttons to control our TVs, navigational systems in our cars, smart phones and other devices switching to smart screen's technology instead. This saved users' time and was more convenient and enjoyable to use.

Times have changed and so did the customer preferences. Nowadays, more and more people don't want to use touch screens. Many of us think it's unhygienic to use a touch screen of a public device such as an informational table, especially in times of the pandemic. For other people like car drivers, touch screens are often unsafe because the former need to distract their attention from driving to perform certain operations with the car's navigation system. Touch screens are also not always convenient to use when you try to click one of its elements that is small in size. Last but not least, touch screens can be touched accidentally when you don't want it to.

Touch less interfaces can solve these and many other problems touch screens have. One important characteristic of touch less interfaces is gesture functionality. In comparison to touch screen devices, touch less devices does not require us to touch the actual screen to control the device. We can instead use gestures and voice commands to control the device. This is more convenient, hygienic, innovative and appealing, especially for millennia’s.

1.5 Application Areas:

1.5.1. Vending Machines:

There were times when many of us have tried to force the vending machine to accept your rumpled dollar bill to buy a bottle of mineral water. Long and unpleasant experience. Was it convenient to pay for your coffee when you needed to take your gloves off and back on to look for coins in your wallet during the cold winter days? Well, those days have come to the end thanks to the intelligent vending machines.

These types of devices are a great example where the hand gesture recognition is being used. Controlling such a device doesn't require interaction with its screen or buttons. All the user has to do is to show specific hand gestures we've discussed in the previous chapter or wave his hand either to the left or right, up or down. Payment can be easily made by simply waving the user's smart phone in front of the reader, solving the problems we've described at the beginning of this chapter.

This solution is especially useful in today's times when customers often don't want to touch the screens of the devices that might have been used by the other individuals before.

1.5.2. Smart Homes:

Apart from implementing the technology in the vending machines, there are other cases where it's also being applied. Take an example of a smart house where hand recognition software allows us to track a hand’s 3D position, rotation and gesture. This allows smart house owners to operate the lights without ever touching the switch, simply by waving their hand in front of it to activate the lights.

1.5.3. Smart TVs:

Another example is the Smart TV. In their case, human face and natural hand gestures are the main components to interact with the smart TV system. The face recognition system is used for identifying the user and the hand gesture recognition controls the actual TV, for example, changing the channels and adjusting the volume.

1.5. 4. Virtual In-Store Displays:

Last but not least, hand gesture recognition is also used in commercial in-store displays which can be found in shopping malls to attract more visitor traffic. Retail business is being increasingly digitized. This includes an introduction of multiple smart devices working together on a single IoT platform to deliver hyper-personalized, adaptive and context-specific experiences. While much of the technology is to be invisible to the consumer, shoppers will have the opportunity to interact digitally within the physical store environment to find out the information they are interested in and sometimes for entertainment purposes.

Chapter 2

LITERATURE SURVEY

2.1 Existed System:

Computer technology has tremendously grown over the past decade and has become a necessary part of everyday live. The primary computer accessory for Human Computer Interaction (HCI) is the mouse.

The mouse is not suitable for HCI in some real life situations, such as with Human Robot Interaction (HRI). There have been many researches on alternative methods to the computer mouse for HCI.

The most natural and intuitive technique for HCI, that is a viable replacement for the computer mouse is with the use of hand gestures.

This project is therefore aimed at investigating and developing a Computer Control (CC) system using hand gestures. Most laptops today are equipped with webcams, which have recently been used insecurity applications utilizing face recognition. In order to harness the full potential of a webcam, it can be used for vision based CC, which would effectively eliminate the need for a computer mouse or mouse pad. The usefulness of a webcam can also be greatly extended to other HCI application such as a sign language database or motion controller.

2.1.1 Drawbacks and Possible solutions:

Gesture-based interfaces have many advantages and provide the user with a completely new form of interaction. However, this kind of input also raises issues that are not relevant with traditional input. On the user's side, these problems are to learn, to remember and to accurately execute gestures. The developer has to provide a system that correctly recognizes these gestures. Freeman et al. remarked that the observation of gestures does not suffice in order to learn them, as the observer is unable to differentiate relevant and irrelevant movements. Therefore, the developer not only has to ensure that gestures are quickly and correctly recognized, but also has to provide a guide that allows a rapid and easy learning of these gestures.

The teaching of multi-touch and mid-air gestures is more difficult than that of single-touch gestures. In the case of the latter, the hand posture is irrelevant - users only need to follow a path correctly to perform a command. But with an extension to multi-touch and mid-air gestures, the position and movement of several fingers or even the whole hand becomes relevant. Teaching systems usually instruct the user about the necessary hand movement and path for a gesture rather than the posture and form of contact, focusing on commands that can also be performed with a single-touch input device like a mouse or a pen.

2.1.1.1 Discoverability:

A disadvantage with gestures is the fact that they are neither self-revealing nor self-explanatory. A named button on a toolbar has an explicit purpose and is also easy to find, gestures, however, may be arbitrary and are usually more difficult to discover.

In order to solve this problem, Bau and Mackay (2008) proposed *OctoPocus*, a dynamic guide that combines feed forward and feedback mechanisms. After a press-and-wait gesture, a map of all possible gestures, visualized through colored templates, is displayed around the current cursor position. As the user begins to follow a path, the other paths become progressively thinner, indicating that they're less likely to be recognized, until they disappear.

The possible gestures Cut, Copy and Paste are displayed around the current cursor position, visualized as colored paths with bolder prefixes. As the user begins to follow a path, the prefixes move accordingly and commands that differ too much from the current path become thinner (Cut) or even disappear (Paste). (Bau & Mackay, 2008)

A solution that is also suitable for multi-touch input is the *Shadow Guides* system by Freeman et al. A so-called user-shadow visualizes the user's input, giving feedback on what parts of the hand are in contact with the surface. The user shadow annotations demonstrate possible gestures available from the current hand pose; the registration pose guide informs the user about alternative registration poses. (2009)

Another, a little different approach is *Gesture Bar* by Bragdon et al. While the aforementioned learning guides employ the "learning-by-doing" technique, *Gesture Bar* separates the learning area from the user's document and discloses information about a gesture only if needed. The system works like a traditional toolbar - the user can click an item to find details about the execution of the command and to test it in an experimental area. (Bragdon, Zeleznik, Williamson, Miller & LaViola, 2009)

2.1.1.2 Memorability:

While conventional commands only have to be recognized, gestures need to be known and remembered before executing those (Bau& Mackay, 2008).

One possibility to create memorable gestures is to make them as intuitive as possible, as they are more likely to be remembered that way (Wachs, Kölsch, Stern & Edan, 2011). Wobbrock et al. researched these natural gestures and found that although there are common features used by nearly all of the participants, gestures are far from being "obvious" and that it is difficult to design a gesture set that feels natural for every user. People often used reversible gestures to achieve two opposing effects and used more fingers for moving larger objects, mirroring their experiences in the real world. They were also strongly influenced by their knowledge of traditional computers, using gestures that could also be performed with a mouse (even tapping their fingers as if clicking it) and locating the "Close" gesture at the top-right corner of objects as if they were using a Windows PC. (2009)

Another aspect to remember is the fact that the concept of intuitiveness strongly depends on culture and experience. Many mid-air gestures used in everyday life strongly differ from country to country - a nod, for example, will be commonly interpreted as an indication of agreement, but there are some countries, like Greece for instance, where it stands for the exact opposite. Another example is the pinch-to-zoom gesture that will come natural to every regular smart phone user, but not to someone who has never seen a touch screen.

An alternative to intuitive gestures are the so-called Marking Menus that combine named commands in a pie menu and gestures. That way, they ease the transition from novice- to expert-mode usage. A further development of Marking Menus are the *Augmented Letters* by Roy et al., where the user activates a pie menu by drawing the letter the elements start with. After that, he can choose from the menu by extending the gesture (see Figure 2). (Roy, Malacria, Guiard, Lecolinet & Eagan, 2013) illustration not visible in this excerpt.

2.1.1.3 Fatigue:

Gestures normally involve more muscles than other interaction techniques (Baudel & Beaudouin-Lafon, 1993), and especially mid-air gestures, but also gestures that require muscle tension and complex movements over a long period of time can be very exhausting. Therefore, developers should design gestures that are quick and comfortable to execute.

One approach is the so-called *Micro gestures* - tiny gestures that can even be executed during other activities and therefore allows true multitasking. A possible area of usage is driving, where small tasks like changing the volume of the radio can thus be performed without the potential risk of releasing the steering wheel. This idea has of course been already partially implemented with the integration of additional control elements in the steering wheel. But micro interactions could also be incorporated in other fields of our everyday life, like when writing with a pen or holding a cash card.

Another aspect, addressed by Forlines and others that need to be considered is the fact that many bimanual interactions in the real world are asymmetric, with the non-dominant hand being slower and less precise than the dominant hand. Therefore, gestures need to be equally easy for left- and right-handed users and should not demand too much of a person's non-dominant hand.

2.2 Proposed System:

For most laptop touchpad is not the most comfortable and convenient way to operate .So we propose Virtual hand tracking, which can be used as a virtual mouse. This is real time application and User friendly application.

- This project removes the requirement of having a physical contact with the device.
- It leads to development in virtual reality, augmented reality and holographic access.

Gesture recognition is a technique which is used to understand and analyze the human body language and interact with the user accordingly. This in turn helps in building a bridge between the machine and the user to communicate with each other. Gesture recognition is useful in processing the information which cannot be conveyed through speech or text. Gestures are the simplest means of communicating something that is meaningful. This paper involves implementation of the system that aims to design a vision-based hand gesture recognition system with a high correct detection rate along with a high-performance criterion, which can work in a real time Human Computer Interaction system without having any of the limitations (gloves, uniform background etc.) on the user environment. The system can be defined using a flowchart that contains three main steps, they are: *Learning, Detection, Recognition.*

i) Learning:
  (1) It involves two aspects such as
    (a) Training dataset: This is the dataset that consists of different types of hand gestures that are used to train the system based on which the system performs the actions.
    (b) Feature Extraction: It involves determining the centroid that divides the image into two halves at its geometric Centre.

ii) Detection
  (1) Capture scene: Captures the images through a web camera, which is used as an input to the system.
  (2) Preprocessing: Images that are captured through the webcam are compared with the dataset to recognize the valid hand movements that are needed to perform the required actions.
  (3) Hand Detection: The requirements for hand detection involve the input image from the webcam. The image should be fetched with a speed of 20 frames per second. Distance should also be maintained between the hand and the camera. Approximate distance that should be between hands the camera is around 30 to 100 cm. The video input is stored frame by frame into a matrix after preprocessing.

iii) Recognition

(1) Gesture Recognition: The number of fingers present in the hand gesture is determined by making use of defect points present in the gesture. The resultant gesture obtained is fed through a 3Dimensional Convolution Neural Network consecutively to recognize the current gesture.

(2) Performing action: The recognized gesture is used as an input to perform the actions required by the user.

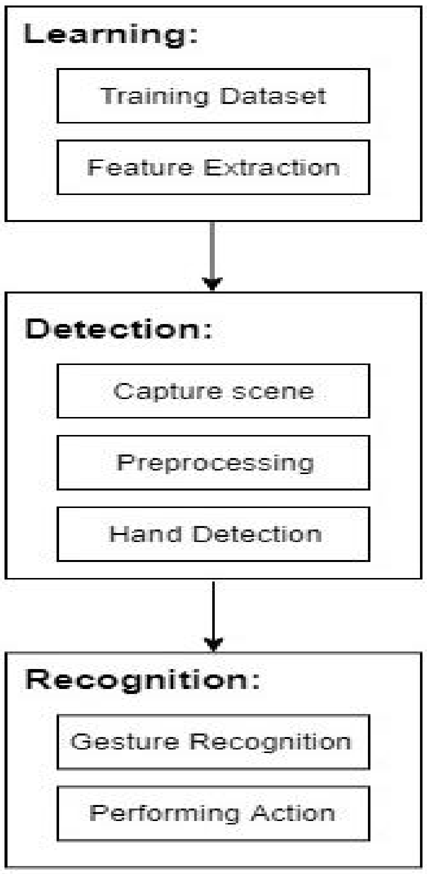


Fig 2.2 Proposed system

Screen 2.2 Proposed System

Chapter 3

REQUIREMENT ANALYSIS

3.1 System Requirements:

3.1.1 Hardware Requirements:

1. PC with Webcam.
2. 4 GB RAM
3. 500 GB Hard disk
4. Intel i5 Processor.

3.1.1.1 RAM:

Random-access memory (**RAM**) is a type of storage for computer systems that makes it possible to access data very quickly in random order.A **4GB RAM means** it can store **4GB** of data temporarily or if you count in number of characters then a **4GB RAM** can hold 4294967296 characters.
RAM is a form of computer memory that can be read and changed in any order, typically used to store working data and machine code. A random-access memory device allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory. In contrast, with other direct-access data storage media such as hard disks, CD-RWs, DVD-RWs and the older magnetic tapes and drum memory, the time required to read and write data items varies significantly depending on their physical locations on the recording medium, due to mechanical limitations such as media rotation speeds and arm movement.

RAM contains multiplexing and demultiplexing circuitry, to connect the data lines to the addressed storage for reading or writing the entry. Usually more than one bit of storage is accessed by the same address, and RAM devices often have multiple data lines and are said to be "8-bit" or "16-bit", etc. devices.

In today's technology, random-access memory takes the form of integrated circuit (IC) chips with MOS (metal-oxide-semiconductor) memory cells. RAM is normally associated with volatile types of memory (such as dynamic random-access memory (DRAM) modules), where stored information is lost if power is removed, although non-volatile RAM has also been developed. Other types of non-volatile memories exist that allow random access for read operations, but either do not allow write operations or have other kinds of limitations on them. These include most types of ROM and a type of flash memory called *NOR-Flash*.

The two main types of volatile random-access semiconductor memory are static random-access memory (SRAM) and dynamic random-access memory (DRAM). Commercial uses of semiconductor RAM date back to 1965, when IBM introduced the SP95 SRAM chip for their System/360 Model 95 computer, and Toshiba used DRAM memory cells for its Toscal BC-1411 electronic calculator, both based on bipolar transistors. Commercial MOS memory, based on MOS transistors, was developed in the late 1960s, and has since been the basis for all commercial semiconductor memory. The first commercial DRAM IC chip, the Intel 1103, was introduced in October 1970. Synchronous dynamic random-access memory (SDRAM) later debuted with the Samsung KM48SL2000 chip in 1992.

3.1.1.2 Intel i5 Processor:

Developed and manufactured by Intel, the **Core i5** is a computer processor, available as dual-core or quad-core. It can be used in both desktop and laptop computers, and is one of four types of processors in the "i" (Intel Core family) series. The first i5 processor was released in September 2009 and new generations of the i5 continue to be released (2020).

The Core i5 processor is available in multiple speeds, ranging from 1.90 GHz up to 3.80 GHz, and it features 3 MB, 4 MB or 6 MB of cache. It utilizes either the LGA 1150 or LGA 1155 socket on a motherboard. Core i5 processors are most often found as quad-core, having four cores. However, a select few high-end Core i5 processors feature six cores.

The most common type of RAM used with a Core i5 processor is DDR3 1333 or DDR3 1600. However, higher performance RAM can be used as well if it's supported by the motherboard.

Power usage varies for the Core i5 processors:

- Slower speeds (1.90 GHz to 2.30 GHz) use 11.5 W of power
- Medium speeds (2.60 GHz to 3.10 GHz) use 15 W, 25 W, 28 W or 37 W of power
- Faster speeds (3.20 GHz to 3.80 GHz) use 35 W, 37 W, 45 W, 47 W, 65 W or 84 W of power

Core i5 processors are commonly found in desktop computers for most everyday use and some higher performance needs. Some laptop computers feature Core i5 processors as well, to provide improved performance for heavier usage needs. At the lower speeds, battery usage is pretty conservative and can reach up to five hours or usage on a single charge. However, at higher speeds, battery usage is higher and may result in up to three hours or so of usage per charge.

3.1.2 Software Requirements:

- ➢ Python
- ➢ Open CV
- ➢ Media Pipe Framework

3.1.2.1 Python:

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a garbage collection system using reference counting. Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible and much Python 2 code does not run unmodified on Python 3. Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages

Python 2.0 was released on 16 October 2000, with many major new features, including a cycle-detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were back ported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. No more security patches or other improvements will be released for it. With Python 2's end-of-life, only Python 3.6.x and later are supported.

Python 3.9.2 and 3.8.8 were expedited as all versions of Python (including 2.7) had security issues, leading to possible remote code execution and web cache poisoning.

3.1.2.2 opencv

OpenCV is a cross-platform library using which we can develop real-time **computer vision applications**. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

Let’s start the chapter by defining the term "Computer Vision".

Computer Vision

Computer Vision can be defined as a discipline that explains how to reconstruct, interrupt, and understand a 3D scene from its 2D images, in terms of the properties of the structure present in the scene. It deals with modeling and replicating human vision using computer software and hardware.

Computer Vision overlaps significantly with the following fields −

- **Image Processing** − It focuses on image manipulation.
- **Pattern Recognition** − It explains various techniques to classify patterns.
- **Photogrammetry** − It is concerned with obtaining accurate measurements from images.

Computer Vision Vs Image Processing

Image processing deals with image-to-image transformation. The input and output of image processing are both images.

Computer vision is the construction of explicit, meaningful descriptions of physical objects from their image. The output of computer vision is a description or an interpretation of structures in 3D scene.

Applications of Computer Vision

Here we have listed down some of major domains where Computer Vision is heavily used.

Robotics Application

- Localization − Determine robot location automatically
- Navigation
- Obstacles avoidance
- Assembly (peg-in-hole, welding, painting)
- Manipulation (e.g. PUMA robot manipulator)
- Human Robot Interaction (HRI) − Intelligent robotics to interact with and serve people

Medicine Application

- Classification and detection (e.g. lesion or cells classification and tumor detection)

- 2D/3D segmentation
- 3D human organ reconstruction (MRI or ultrasound)
- Vision-guided robotics surgery

Industrial Automation Application

- Industrial inspection (defect detection)
- Assembly
- Barcode and package label reading
- Object sorting
- Document understanding (e.g. OCR)

Security Application

- Biometrics (iris, finger print, face recognition)
- Surveillance − Detecting certain suspicious activities or behaviors

Transportation Application

- Autonomous vehicle
- Safety, e.g., driver vigilance monitoring

Features of OpenCV Library

Using OpenCV library, you can −

- Read and write images
- Capture and save videos
- Process images (filter, transform)
- Perform feature detection
- Detect specific objects such as faces, eyes, cars, in the videos or images.
- Analyze the video, i.e., estimate the motion in it, subtract the background, and track objects in it.

OpenCV was originally developed in C++. In addition to it, Python and Java bindings were provided. OpenCV runs on various Operating Systems such as windows, Linux, OSx, FreeBSD, Net BSD, Open BSD, etc.

This tutorial explains the concepts of OpenCV with examples using Java bindings.

OpenCV Library Modules

Following are the main library modules of the OpenCV library.

Core Functionality

This module covers the basic data structures such as Scalar, Point, Range, etc., that are used to build OpenCV applications. In addition to these, it also includes the multidimensional array **Mat**, which is used to store the images. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.core**.

Image Processing

This module covers various image processing operations such as image filtering, geometrical image transformations, color space conversion, histograms, etc. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.imgproc**.

Video

This module covers the video analysis concepts such as motion estimation, background subtraction, and object tracking. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.video**.

Video I/O

This module explains the video capturing and video codecs using OpenCV library. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.videoio**.

calib3d

This module includes algorithms regarding basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence and elements of 3D reconstruction. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.calib3d**.

features2d

This module includes the concepts of feature detection and description. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.features2d**.

Objdetect

This module includes the detection of objects and instances of the predefined classes such as faces, eyes, mugs, people, cars, etc. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.objdetect**.

Highgui

This is an easy-to-use interface with simple UI capabilities. In the Java library of OpenCV, the features of this module is included in two different packages namely, **org.opencv.imgcodecs** and **org.opencv.videoio**.

A Brief History of OpenCV

OpenCV was initially an Intel research initiative to advise CPU-intensive applications. It was officially launched in 1999.

- In the year 2006, its first major version, OpenCV 1.0 was released.
- In October 2009, the second major version, OpenCV 2 was released.
- In August 2012, OpenCV was taken by a nonprofit organization OpenCV.org.

3.1.2.3 Media pipe framework:

The ability to perceive the shape and motion of hands can be a vital component in improving the user experience across a variety of technological domains and platforms. For example, it can form the basis for sign language understanding and hand gesture control, and can also enable the overlay of digital content and information on top of the physical world in augmented reality. While coming naturally to people, robust real-time hand perception is a decidedly challenging computer vision task, as hands often occlude themselves or each other (e.g. finger/palm occlusions and hand shakes) and lack high contrast patterns.

Today we are announcing the release of a new approach to hand perception, which we previewed CVPR 2019 in June, implemented in MediaPipe—an open source cross platform framework for building pipelines to process perceptual data of different modalities, such as video and audio. This approach provides high-fidelity hand and finger tracking by employing machine learning (ML) to infer 21 3D keypoints of a hand from just a single frame. Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, our method achieves real-time performance on a mobile phone, and even scales to multiple hands. We hope that providing this hand perception functionality to the wider research and development community will result in an emergence of creative use cases, stimulating new applications and new research avenues.

An ML Pipeline for Tracking and Gesture Recognition Hand

Our hand tracking solution utilizes an ML pipeline consisting of several models working together:

- A palm detector model (called BlazePalm) that operates on the full image and returns an oriented hand bounding box.
- A hand landmark model that operates on the cropped image region defined by the palm detector and returns high fidelity 3D hand keypoints.
- A gesture recognizer that classifies the previously computed keypoint configuration into a discrete set of gestures.

This architecture is similar to that employed by our recently published face mesh ML pipeline and that others have used for pose estimation. Providing the accurately cropped palm image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy.

BlazePalm: Realtime Hand/Palm Detection

To detect initial hand locations, we employ a single-shot detector model called BlazePalm, optimized for mobile real-time uses in a manner similar to BlazeFace, which is also available in MediaPipe. Detecting hands is a decidedly complex task: our model has to work across a variety of hand sizes with a large scale span (~20x) relative to the image frame and be able to detect occluded and self-occluded hands. Whereas faces have high contrast patterns, e.g., in the eye and mouth region, the lack of such features in hands makes it comparatively difficult to detect them reliably from their visual features alone. Instead, providing additional context, like arm, body, or person features, aids accurate hand localization.

Our solution addresses the above challenges using different strategies. First, we train a palm detector instead of a hand detector, since estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. In addition, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases, like handshakes. Moreover, palms can be modelled using square bounding boxes (anchors in ML terminology) ignoring other aspect ratios, and therefore reducing the number of anchors by a factor of 3-5. Second, an encoder-decoder feature extractor is used for bigger scene context awareness even for small objects (similar to the RetinaNet approach). Lastly, we minimize the focal loss during training to support a large amount of anchors resulting from the high scale variance.

With the above techniques, we achieve an average precision of 95.7% in palm detection. Using a regular cross entropy loss and no decoder gives a baseline of just 86.22%.

Hand Landmark Model

After the palm detection over the whole image our subsequent hand landmark model performs precise keypoint localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, that is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

To obtain ground truth data, we have manually annotated ~30K real-world images with 21 3D coordinates, as shown below (we take Z-value from image depth map, if it exists per corresponding coordinate). To better cover the possible hand poses and provide additional supervision on the nature of hand geometry, we also render a high-quality synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates.

Chapter 4
DESIGN

4.1 Hand Gestures for Computer Vision:

Gestures are expressive, meaningful body motions with the intent to convey information or interact with the environment. According to Cadoz hand gestures serve three functional roles, semiotic, ergotic, and epistemic. The semiotic function is to communicate information, the ergotic function corresponds to the capacity to manipulate objects in the real world, and the epistemic function allows us to learn from the environment through tactile experience. Based on this classification Quek distinguishes communicative gestures, which are meant for visual interpretation and where no hidden part carries information critical to understanding, from manipulative gestures, which show no such constraints. Thus, it may be more appropriate to use special tools for interaction, like data gloves, rather than computer vision if the intent is realistic manipulation of objects in, e.g., a virtual environment. Pavlovic et al. makes a similar classification, but also point out the distinction between unintentional movements and gestures.

For communicative, semiotic gestures, Kendon distinguishes gesticulation, gestures that accompany speech pantomimes, emblems, and sign languages. When moving forward in this list the association with speech diminishes, language properties increase, spontaneity decreases and social regulation increases. Detailed descriptions and taxonomies concerning hand gestures from the point of view of computer vision can be found in Quek, Pavlovic & Sharma and Turk. Here only a brief overview will, from autonomous gestures. These can be of four different kinds: language-like gestures, be presented. Most work in computer vision and HCI has focused on emblems and signs because they carry more clear semantic meaning, and may be more appropriate for command and control interaction. It is important to note, however, that they are largely symbolic, arbitrary in nature, and that universally understandable gestures of this kind hardly exist. There is also one important exception worth mentioning. In the gesticulation category, McNeill defines deictic gestures as pointing gestures that refer to people, objects, or events in space and time. Deictic gestures are potentially useful for all kinds of selections in human computer interaction, as illustrated, e.g., by the early work of Bolt. The deictic category itself can be further subdivided, but from a computer vision point of view all deictic gestures are performed as pointing, and the difference lies in the higher level of interpretation. In the following we limit ourselves to intentional, semiotic, hand gestures. From a computer vision point of view, we focus on the recognition of static postures and gestures involving movements of fingers, hands and arm with the intent to convey information to the environment.

4.2 Hand Gesture Recognition:

Gesture recognition is a technology aimed at providing live-time data to a computer to execute commands the user wants. People do not need to type anything with keys or tap on a touch screen to

perform a specific action. The device's motion sensor can perceive and interpret the person's movements as the primary source of data input.
We has developed its hand tracking based on neural networks which consist of a stack of two deep-learning technologies. The first of them is a neural network which is trained and configured to identify hands in images taken with 2D mobile device cameras. The second part is a technology that precisely detects 11-21 hand points applied for the real-time precise detection of 11-21 hand points and further hand tracking. Both networks are trained on manually annotated datasets. At runtime, the input frames are scaled-down and sent to the first type of network which detects and localizes a user’s hand. If the hand is detected, the Skeleton Model is activated to predict key points and track position and attitude of the hand. Additional regression-based models may be applied to predict gesturing, e.g. fist or palm. The detected gestures may be used as a part of the user interface in mobile applications.
Below is a step-by-step description of how Banuba's gesture recognition algorithm works:

- **Step 1:** At the first stage, the neural network responsible for identifying hands in images is used to detect a user’s hand in a region of interest. If the hand is detected, the Encoder neural network is activated to predict key points.
- **Step 2:** A separate algorithm detects the position and attitude of the hand and regression model are utilized to predict gestures, e.g. fist or palm.
- **Step 3.** Finally, the detected gestures are used as application control elements. The gestures performed by a person are compared to the gesture library stored in the computer and once the match is found, the computer executes the command correlated to that specific gesture.

Our hand tracking system is capable of recognizing the most common gestures such as:

- Palm ✋
- Victory ✌
- Rock 🤘
- Like 👍
- OK 👌

Now that we've considered what hand gesture recognition is and how it works, let us next look at the few examples of the cases where this technology has been applied.

Hand gestures are an aspect of body language that can be conveyed through the center of thePalm, the finger position and the shape constructed by the hand. Hand gestures can be classified into static and dynamic. As its name implies, the static gesture refers to the stable shape of the hand, whereas the dynamic gesture comprises a series of hand movements such as waving. There are a variety of hand movements within a gesture; for example, a handshake varies from one person to another and changes according to time and place. The main difference between posture and gesture is that posture focuses more on the shape of the hand whereas gesture focuses on the hand movement.

The main approaches to hand gesture research can be classified into the wearable glove-based sensor approach and the camera vision-based sensor approach. Hand gestures over an inspiring field of research because they can facilitate communication and provide a natural means of interaction that can be used across a variety of applications. Previously, hand gesture recognition was achieved with wearable sensors attached directly to the hand with gloves. These sensors detected a physical response according to hand movements or finger bending. The data collected were then processed using a computer connected to the glove with wire. This system of glove-based sensor could be made portable by using a sensor attached to a microcontroller.

As illustrated in Figure hand gestures for human–computer interaction (HCI) started with the invention of the data glove sensor. It ordered simple commands for a computer interface. The gloves used different sensor types to capture hand motion and position by detecting the correct coordinates of the location of the palm and fingers. Various sensors using the same technique based on the angle of bending were the curvature sensors, angular displacement sensor, optical fiber transducer, flex sensors and accelerometer sensor. These sensors exploit different physical principles according to their type. Require cumbersome gloves to be worn. These techniques are called camera vision-based sensor technologies. With the evolution of open-source software libraries, it is easier than ever to detect hand gestures that can be used under a wide range of applications like clinical operations, sign language, robot control, virtual environments, home automation, personal computer and tablet, gaming. These techniques essentially involve replacement of the instrumented glove with a camera. Different types of camera are used for this purpose, such as RGB camera, time of flight (TOF) camera, thermal cameras or night vision cameras. Algorithms have been developed based on computer vision methods to detect hands using these different types of cameras. The algorithms attempt to segment and detect hand features such as skin color, appearance, motion, skeleton, depth, 3D model, deep learn detection and more. These methods involve several challenges, which are discussed in this paper in the following sections.

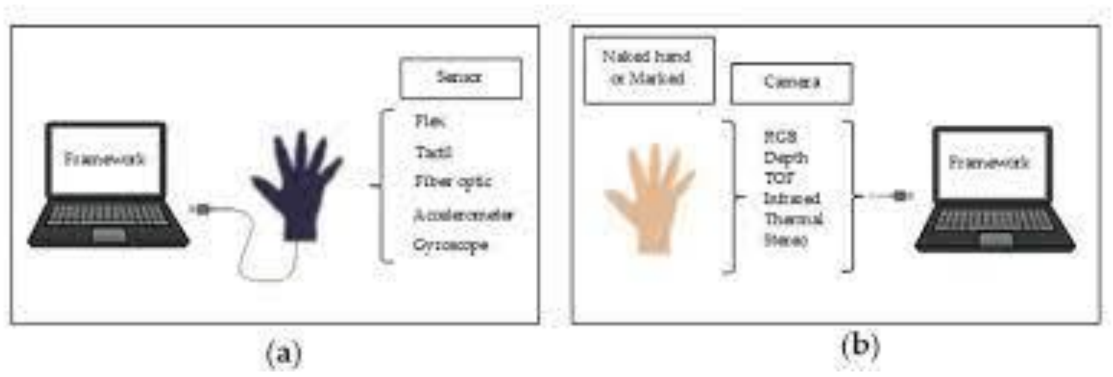


Figure 4.2. Different techniques for hand gestures. 4.2. (**a**) Glove-based attached sensor either connected to the computer or portable; 4.2. (**b**) Computer vision–based camera using a marked glove or just a naked hand.

Although the techniques mentioned above have provided good outcomes, they have various limitations that make them unsuitable for the elderly, who may experience discomfort and confusion due to wire connection problems. In addition, elderly people suffering from chronic disease conditions that result in loss of muscle function may be unable to wear and take of gloves, causing them discomfort and constraining them if used for long periods. These sensors may also cause skin damage, infection or adverse reactions in people with sensitive skin or those suffering burns. Moreover, some sensors are quite expensive. Some of these problems were addressed in a study by Lamberti and Camastra, who developed a computer vision system based on colored marked gloves. Although this study did not require the attachment of sensors; it still required colored gloves to be worn.

These drawbacks led to the development of promising and cost-effective techniques that did not require cumbersome gloves to be worn. These techniques are called camera vision-based sensor technologies. With the evolution of open-source software libraries, it is easier than ever to detect hand gestures that can be used under a wide range of applications like clinical operations, sign language, robot control, virtual environments, home automation, personal computer and tablet, gaming. These techniques essentially involve replacement of the instrumented glove with a camera. Different types of camera are used for this purpose, such as RGB camera, time of flight (TOF) camera, thermal cameras or night vision cameras.

Algorithms have been developed based on computer vision methods to detect hands using these different types of cameras. The algorithms attempt to segment and detect hand features such as skin color, appearance, motion, skeleton, depth, 3D model, deep learn detection and more.

4.3 Perceptive and Multimodal User Interfaces:

The aim is to develop conversational interfaces, based on what is considered to be natural human-to human dialog. For example, Bolt suggested that in order to realize conversational computer interfaces, gesture recognition will have to pick up on unintended gestures, and interpret fidgeting and other body language signs, and Wexelblatt argued that only the use of natural hand gestures is motivated, and that there might even be added cognitive load on the user by using gestures in any other way. Two main scenarios for gestural interfaces can be distinguished. One aims at developing Perceptive User Interfaces (PUI), as described by Turkor Perceptive Spaces, e.g., Wren, striving for automatic recognition of natural, human gestures integrated with other human expressions, such as body movements, gaze, facial expression, and speech.

However, in this paper the focus is on using hand gestures given purposefully as instructions, and we restrict our work to deliberate, expressive movements. This falls within the second approach to gestural interfaces, Multimodal User Interfaces, where hand poses and specific gestures are used as commands in a command language. The gestures need not be natural gestures but could be developed for the situation, or based on a standard sign language. In this approach, gestures are either a replacement for other interaction tools, such as remote controls and mice, or a complement, e.g., gestures used with speech and gaze input in a multimodal interface. Oviatt et al. noted that there is a growing interest in designing multimodal interfaces that incorporate vision-based technologies. They also contrast the passive mode of PUI with the active input mode, addressed here, and claim that although passive modes may be less obtrusive, active modes generally are more reliable indicators of user intent, and not as prone to error.

4.4 Development model:

4.4.1 Waterfall Model

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. The Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

The methodology used for developing the proposed system is waterfall model.

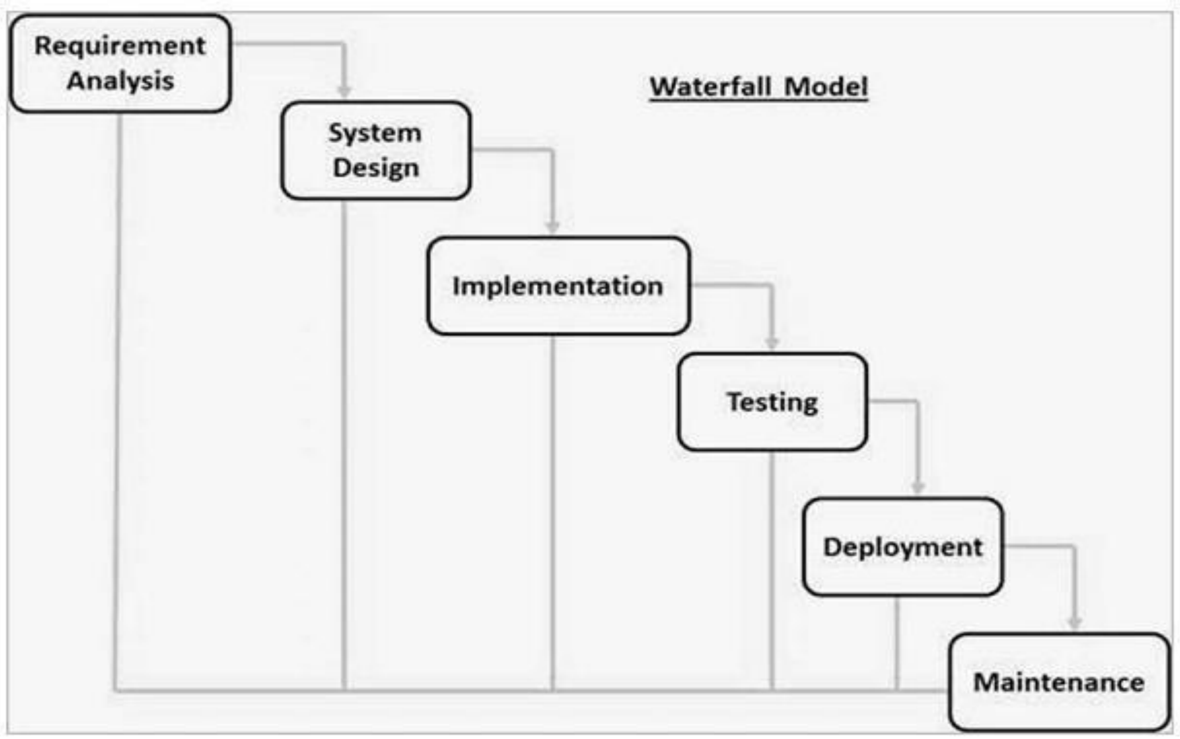


Fig 4.4.1 Water Fall model

The sequential phases in Waterfall model are −

Requirement Gathering and analysis − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
System Design − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
Implementation − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
Integration and Testing − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
Deployment of system − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
Maintenance − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and changes in the customer environment.
All these phases are cascaded to each it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

4.4.2 UNIFIED MODIFIED LANGUAGE (UML) DIAGRAMS:

To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating.
Only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML, there are five diagrams available to model the dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature, there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. Use case diagrams consists of actors,use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system. Hence to model the entire system, a number of use case diagrams are used.
The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and Statechart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified. When the initial task is complete, use case diagrams are modelled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows –

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements are actors

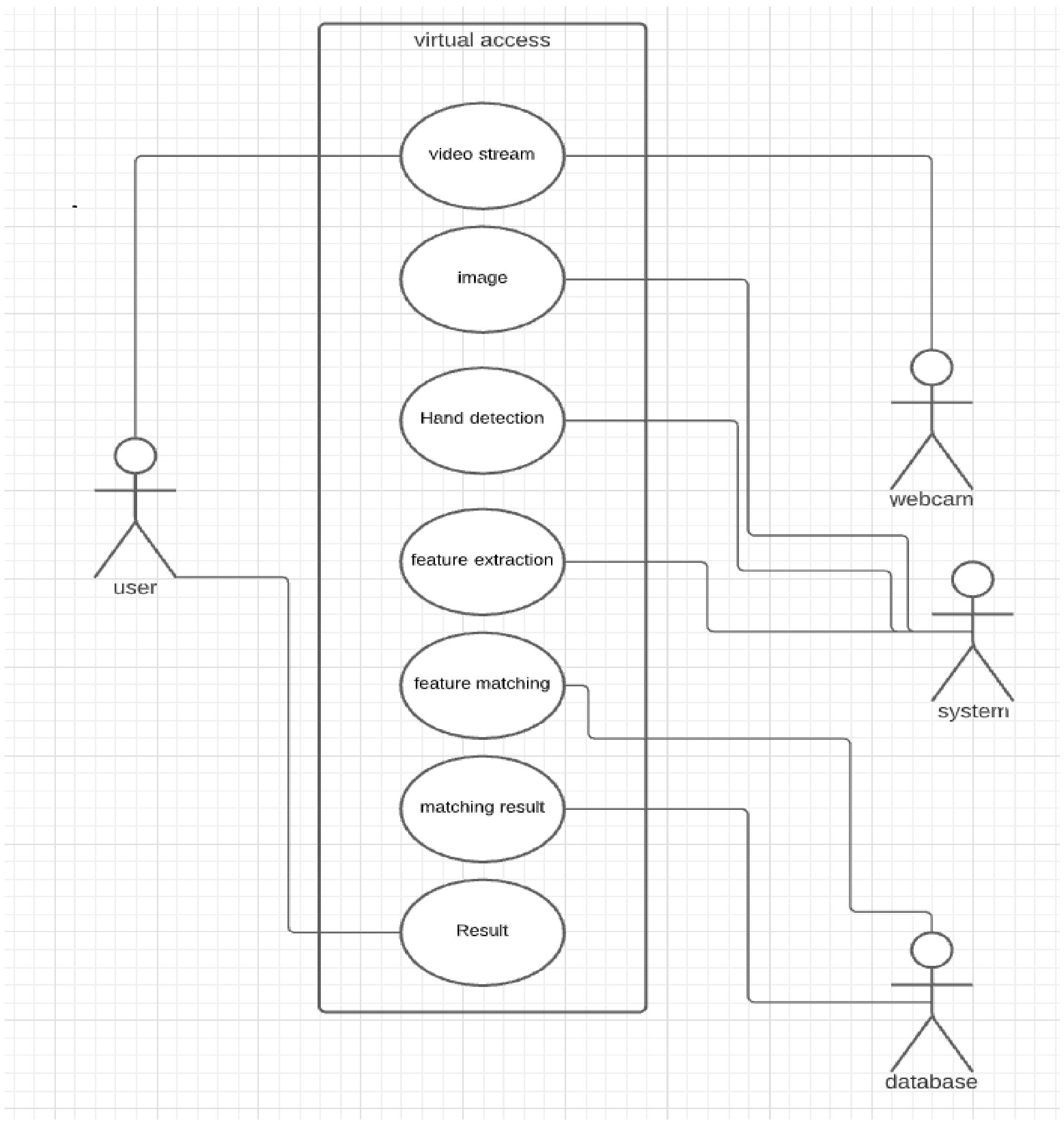


Fig 4.4.2 Use Case

4.4.3 Sequence Diagram

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:

- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.

A **sequence diagram** shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in theorder in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

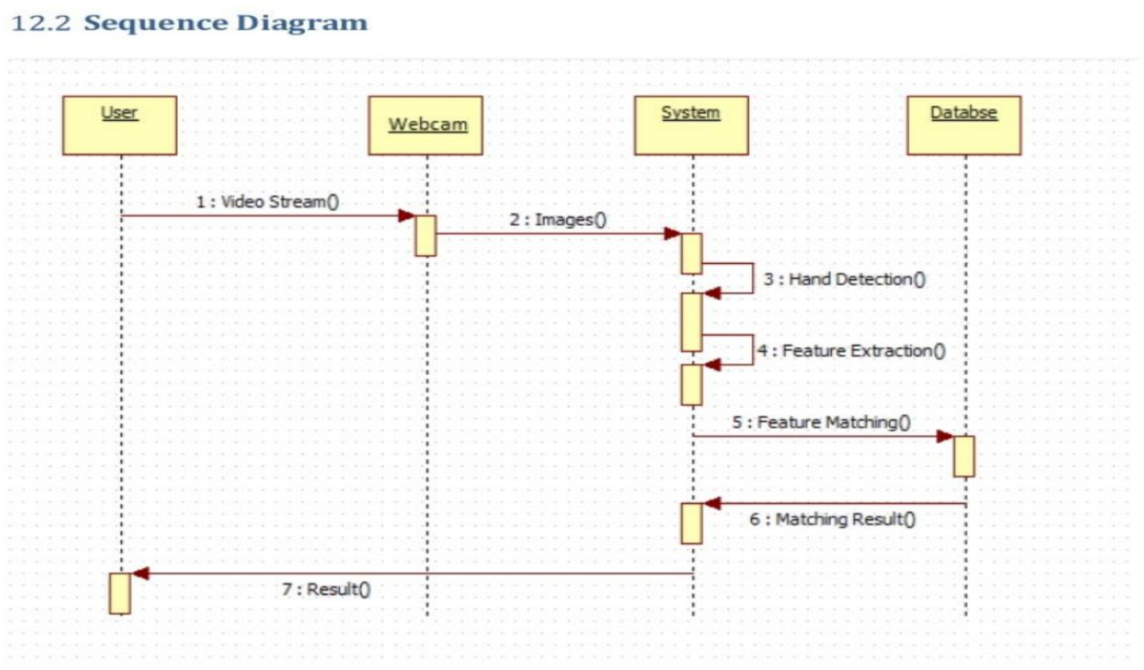


Fig 4.4.3 Sequence Diagram

Chapter 5
IMPLEMENTATION

5.1 System architecture:

5.1.1 Media Pipe:

Media Pipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame. Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, our method achieves real-time performance on a mobile phone, and even scales to multiple hands. We hope that providing this hand perception functionality to the wider research and development community will result in an emergence of creative use cases, stimulating new applications and new research avenue.
Media Pipe Hands utilizes an ML pipeline consisting of multiple models working together:

- A palm detection model that operates on the full image and returns an oriented hand bounding box
- A hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand key points

Providing the accurately cropped hand image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy. In addition, in our pipeline the crops can also be generated based on the hand landmarks identified in the previous frame, and only when the landmark model could no longer identify hand presence is palm detection invoked to re localize the hand.

5.1.2 Palm Detection Model:

To detect initial hand locations, we designed a single-shot detector model optimized for mobile real-time uses. First, we train a palm detector instead of a hand detector, since estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. In addition, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases, like handshakes. Moreover, palms can be modeled using square bounding boxes (anchors in ML terminology) ignoring other aspect ratios, and therefore reducing the number of anchors by a factor of 3-5. Second, an encoder-decoder feature extractor is used for bigger scene context awareness even for small objects. Lastly, we minimize the focal loss during training to support a large amount of anchors resulting from the high scale variance. With the above techniques, we achieve an average precision of 95.7% in palm detection.

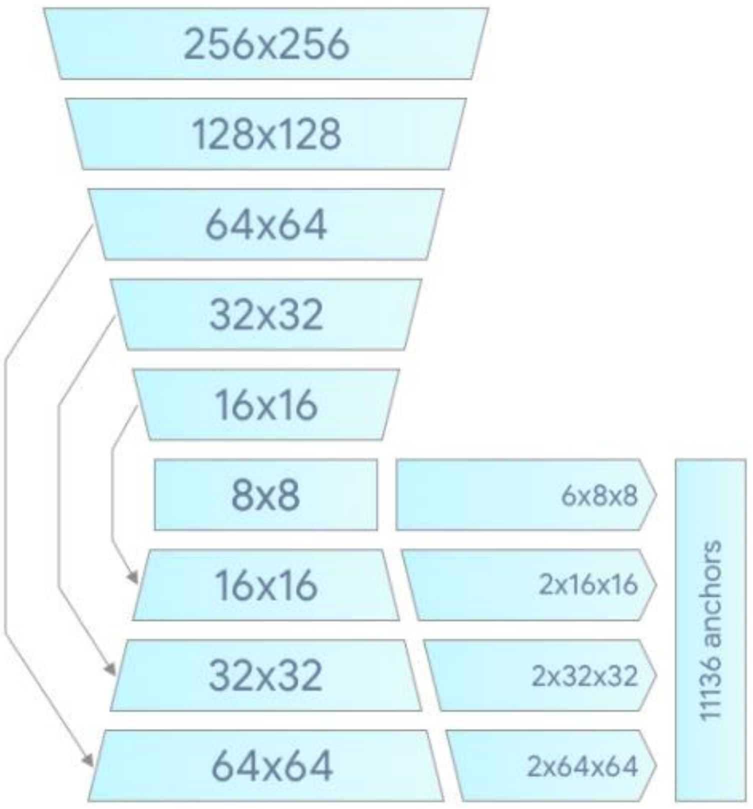


Fig 5.1.2 Palm Detection Model

5.1.3 Hand Land Mark Model:

After the palm detection over the whole image our subsequent hand landmark model performs precise key point localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression that is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

To obtain ground truth data, we have manually annotated ~30K real-world images with 21 3D coordinates, as shown below (we take Z-value from image depth map, if it exists per corresponding coordinate). To better cover the possible hand poses and provide additional supervision on the nature of hand geometry, we also render a high-quality synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates.

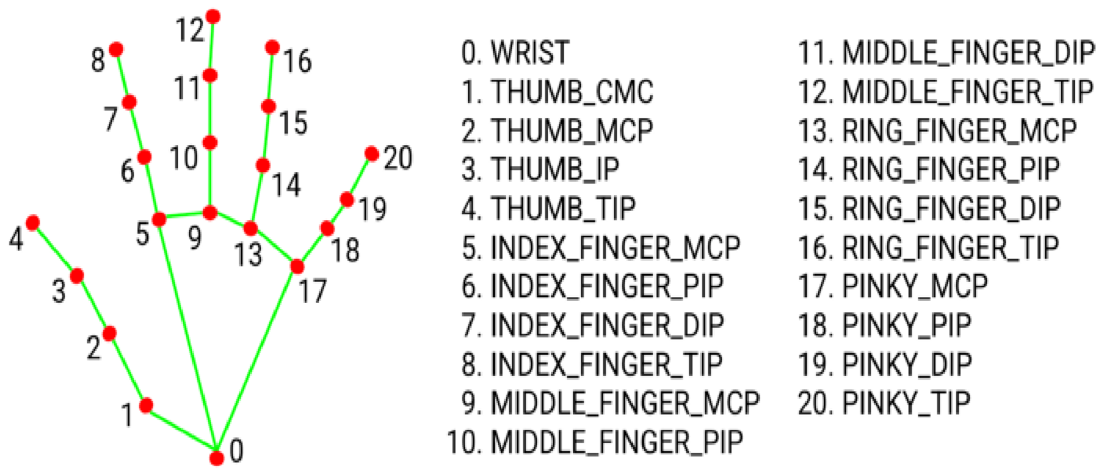


Fig 5.1.3 a) Hand Land Mark Model

The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions. The model has three outputs

- 21 hand landmarks consisting of x, y, and relative depth.
- A hand flag indicating the probability of hand presence in the input image.
- A binary classification of handedness, e.g. left or right hand.

The 2D coordinates are learned from both real-world images as well as synthetic datasets as discussed below, with the relative depth w.r.t. the wrist point being learned only from synthetic images. To recover from tracking failure, we developed another output of the model for producing the probability of the event that a reasonably aligned hand is indeed present in the provided crop. If the score is lower than a threshold, then the detector is triggered to reset tracking.
Handedness is another important attribute for effective interaction using hands in AR/VR. This is especially useful for some applications where each hand is associated with a unique functionality. Thus we developed a binary classification head to predict whether the input hand is the left or right hand. Our setup targets real-time mobile GPU inference, but we have also designed lighter and heavier versions of the model to address CPU inference on the mobile devices lacking proper GPU support and higher accuracy requirements of accuracy to run on desktop, respectively.

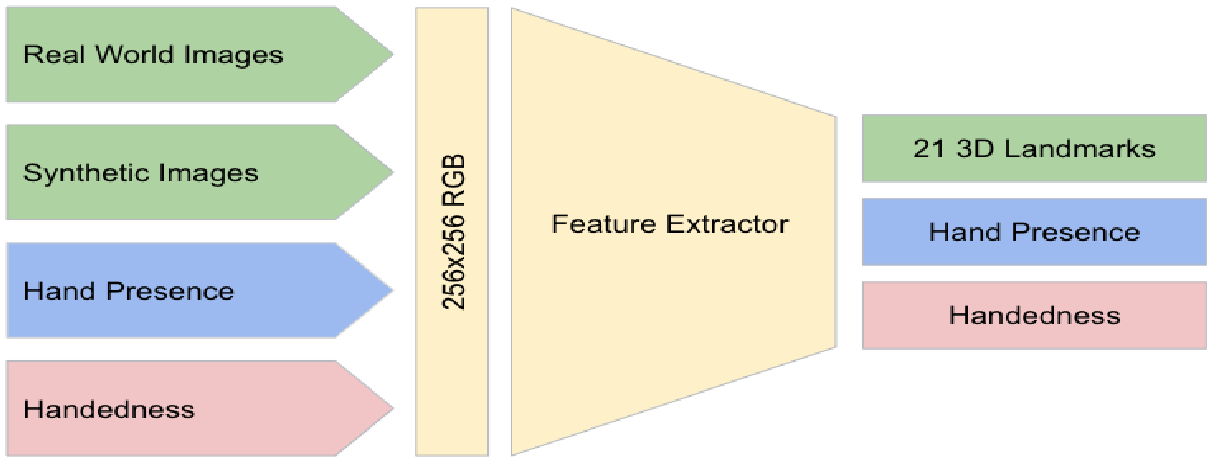


Fig 5.1.3 b) Architecture of Hand Land Mark Model

5.2 Implementation Using Media Pipe:

With Media Pipe, our hand tracking pipeline can be built as a directed graph of modular components, called Calculators. Media pipe comes with an extensible set of Calculators to solve tasks like model inference, media processing, and data transformations across a wide variety of devices and platforms.

Our Media Pipe graph for hand tracking is shown in Figure. The graph consists of two sub graphs one for hand detection and another for landmarks computation. One key optimization Media Pipe provides is that the palm detector only runs as needed (fairly infrequently), saving significant computation. We achieve this by deriving the hand location. In the current video frames from the computed hand landmarks in the previous frame, eliminating the need to apply the palm detector on every frame. For robustness, the hand tracker model also outputs an additional scalar capturing the confidence that a hand is present and reasonably aligned in the input crop. Only when the confidence falls below a certain threshold is the hand detection model reapplied to the next frame.

With Media Pipe, our hand tracking pipeline can be built as a directed graph of modular components, called Calculators. Media pipe comes with an extensible set of Calculators to solve tasks like model inference, media processing, and data transformations across a wide variety of devices and platforms. Individual Calculators like cropping, rendering and neural network computations are further optimized to utilize GPU acceleration.

For example, we employ TFLite GPU inference on most modern phones.
One key optimization Media Pipe provides is that the palm detector only runs as needed (fairly infrequently), saving significant computation. We achieve this by deriving the hand location in the current video frames from the computed hand landmarks in the previous frame, eliminating the need to apply the palm detector on every frame. For robustness, the hand tracker model also outputs an additional scalar capturing the confidence that a hand is present and reasonably aligned in the input crop. Only when the confidence falls below a certain threshold is the hand detection model reapplied to the next frame.

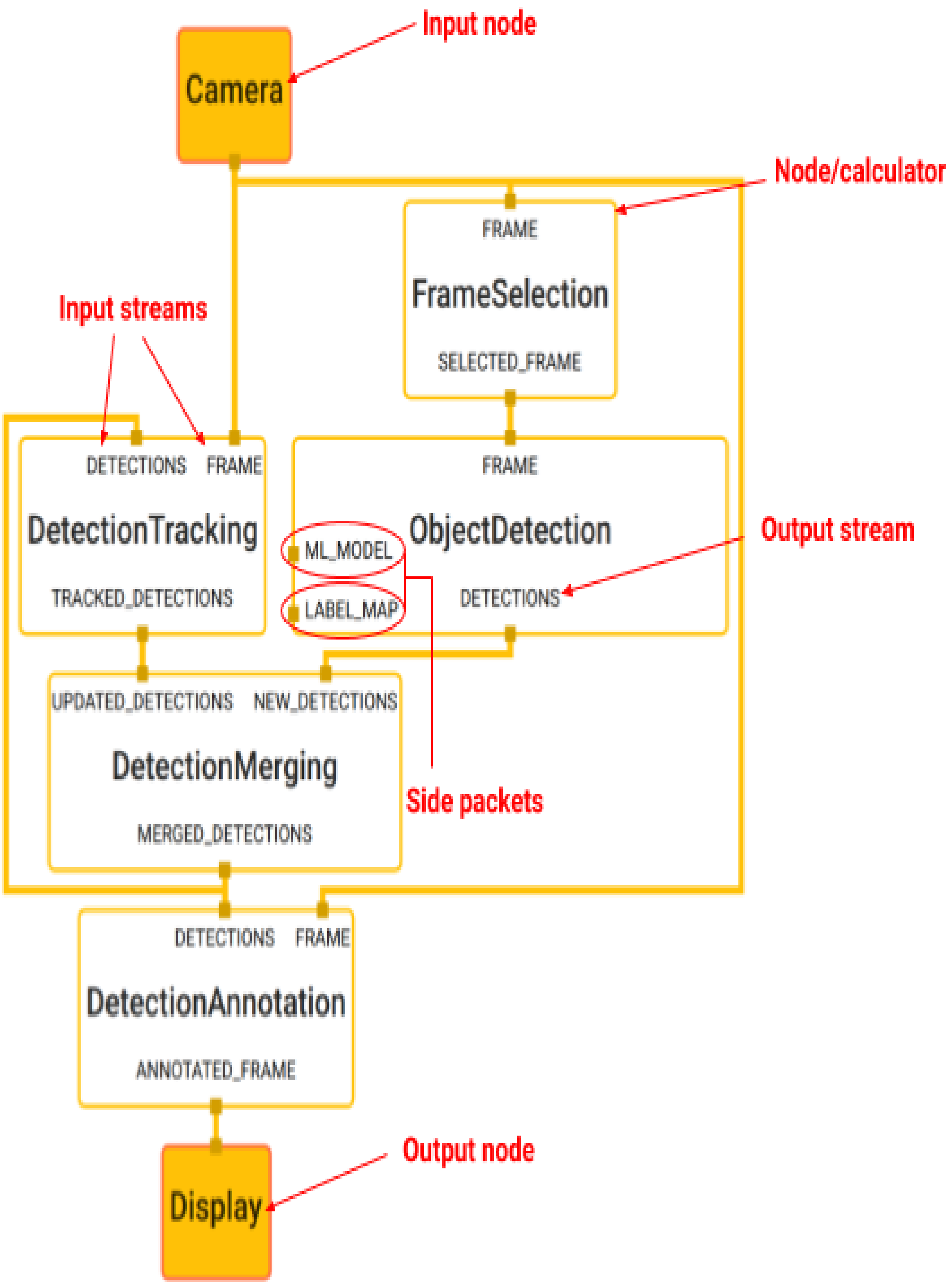


Fig: 5.2 Implementation using Media Pipe

5.3 Flow chart:

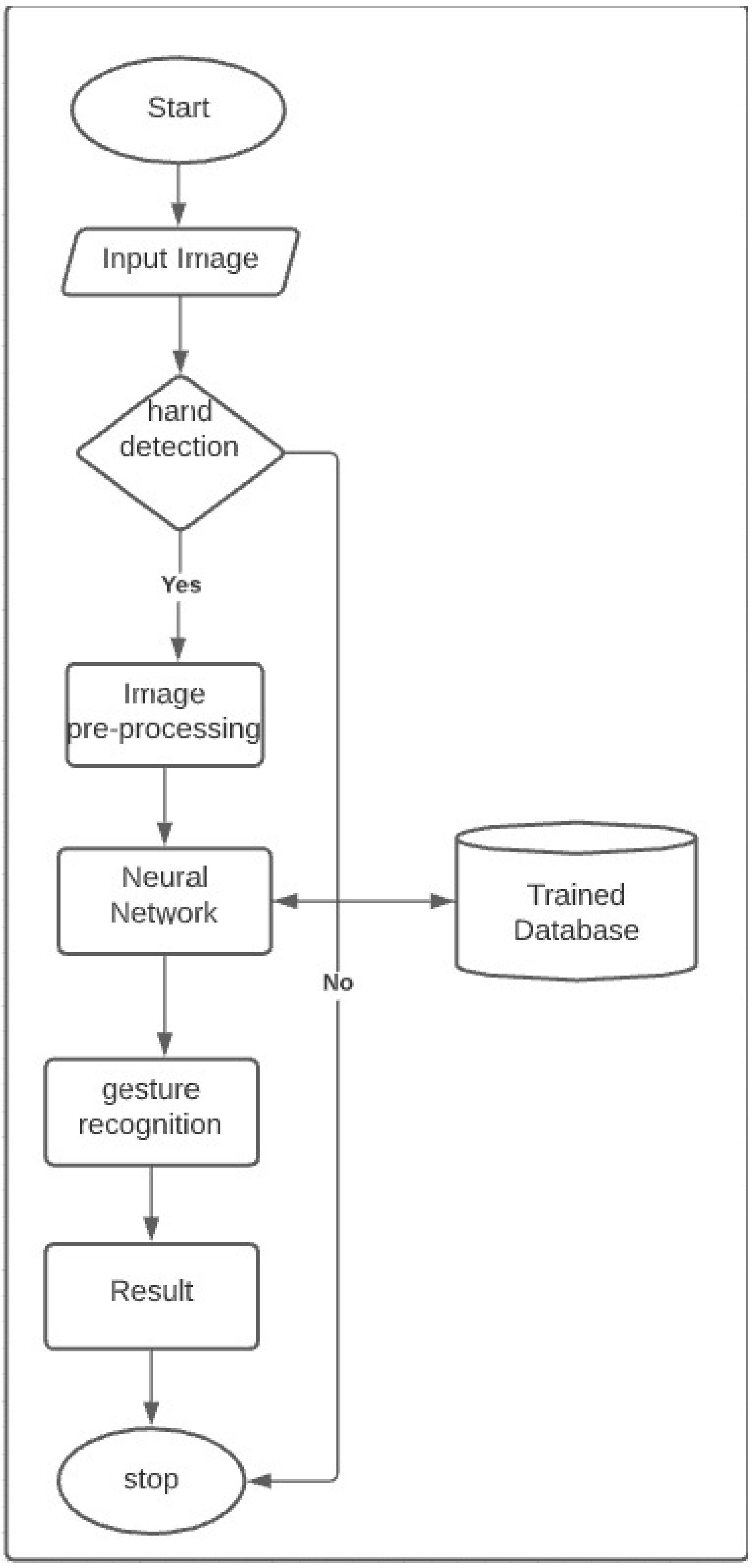


Fig 5.3 Flow Chart

Chapter: 6
SOFTWARE

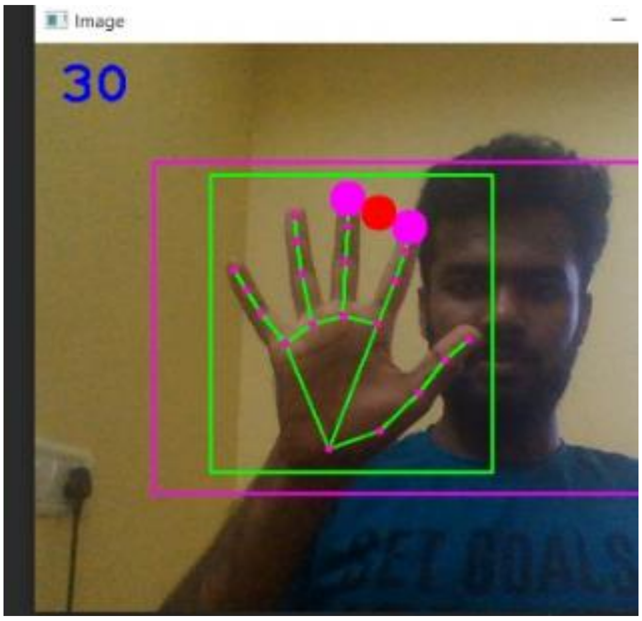
Coding:

```
import cv2
import mediapipe as mp
import time
import math
import numpy as np

“”” Hand detection algorithm, “””
class handDetector():
    def __init__(self,mode=False,maxHands = 2,detectionCon = 0.5, trackCon = 0.5 ):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.trackCon = trackCon

        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands, self.detectionCon,
self.trackCon)

        self.mpDraw = mp.solutions.drawing_utils
        self.tipIds = [4, 8, 12, 16, 20]

    def findHands(self, img, draw = True):
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)

        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms,
self.mpHands.HAND_CONNECTIONS)
        return img

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

        xList = []
        yList = []
        bbox = []
        self.lmList = []
        if self.results.multi_hand_landmarks:
            myHand = self.results.multi_hand_landmarks[handNo]
            for id, lm, in enumerate(myHand.landmark):
                #print(id, lm)
                h, w, c = img.shape
                cx, cy = int(lm.x*w), int(lm.y*h)
                xList.append(cx)
                yList.append(cy)
                #print(id, cx, cy)
                self.lmList.append([id, cx, cy])
                if draw:
                    cv2.circle(img,(cx,cy),3,(255,0,255),cv2.FILLED)

            xmin, xmax = min(xList), max(xList)
            ymin, ymax = min(yList), max(yList)
            bbox = xmin, ymin, xmax, ymax

            if draw:
                cv2.rectangle(img, (xmin - 20, ymin - 20), (xmax + 20, ymax + 20),
                              (0, 255, 0), 2)

        return self.lmList, bbox

    def fingersUp(self):
        fingers = []
        # Thumb
        if self.lmList[self.tipIds[0]][1] > self.lmList[self.tipIds[0] - 1][1]:
            fingers.append(1)
        else:
            fingers.append(0)

        # Fingers
        for id in range(1, 5):

            if self.lmList[self.tipIds[id]][2] < self.lmList[self.tipIds[id] - 2][2]:
                fingers.append(1)
            else:
                fingers.append(0)

        # totalFingers = fingers.count(1)

        return fingers
    def findDistance(self, p1, p2, img, draw=True,r=15, t=3):
        x1, y1 = self.lmList[p1][1:]
        x2, y2 = self.lmList[p2][1:]
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

        if draw:
            cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), t)
            cv2.circle(img, (x1, y1), r, (255, 0, 255), cv2.FILLED)
            cv2.circle(img, (x2, y2), r, (255, 0, 255), cv2.FILLED)
            cv2.circle(img, (cx, cy), r, (0, 0, 255), cv2.FILLED)
        length = math.hypot(x2 - x1, y2 - y1)

        return length, img, [x1, y1, x2, y2, cx, cy]


def main():
    pTime = 0
    cTime = 0
    cap = cv2.VideoCapture(0)
    detector = handDetector()
    while True:
        success, img = cap.read()
        img = detector.findHands(img)
        lmList = detector.findPosition(img)
        if len(lmList) != 0:
            print(lmList[4])
        cTime = time.time()
        fps = 1 / (cTime - pTime)
        pTime = cTime

        cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0,
0), 3)

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

if __name__ == "__main__":
    main()
import cv2
import numpy as np
import handtracking as htm
import time
import autopy
```

```
wCam, hCam = 640, 480
frameR = 100
smoothening =7
plocx,plocy = 0,0
clocx,clocy = 0,0

cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)

pTime = 0
cTime = 0

detector = htm.handDetector(maxHands=1)
wScr, hScr = autopy.screen.size()
#print(wScr, hScr)

while True:
    success, img = cap.read()
    img = detector.findHands(img)
    lmList, bbox =  detector.findPosition(img)

    if len(lmList)!= 0:
        x1, y1 = lmList[8][1:]
        x2, y2 = lmList[12][1:]
        #print(x1, y1, x2, y2)

        finges = detector.fingersUp()
        #print(finges)
        cv2.rectangle(img, (frameR, frameR), (wCam - frameR, hCam - frameR), (255, 0,
255), 2)
        if finges[1]==1 and finges[2]==0:

            x3 = np.interp(x1, (frameR,wCam-frameR), (0, wScr))
            y3 = np.interp(y1, (frameR, hCam-frameR), (0, hScr))

            clocx = plocx+(x3 - plocx)/smoothening
            clocy = plocy+(y3 - plocy)/smoothening

            autopy.mouse.move(wScr-x3, y3)
            cv2.circle(img,(x1, y1), 10, (255,0,0),cv2.FILLED)
            plocx,plocy = clocx,clocy
        if finges[1]==1 and finges[2]==1:
            length, img,lineInfo =detector.findDistance(8,12,img)
            print(length)
            if length <40:
                cv2.circle(img,(lineInfo[4], lineInfo[5]), 10,
(0,255,0),cv2.FILLED)
                autopy.mouse.click()
    cTime = time.time()
    fps = 1/(cTime-pTime)
    pTime=cTime
    cv2.putText(img,str(int(fps)),(20,50),cv2.FONT_HERSHEY_PLAIN,3
,(255, 0,0),3)
    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

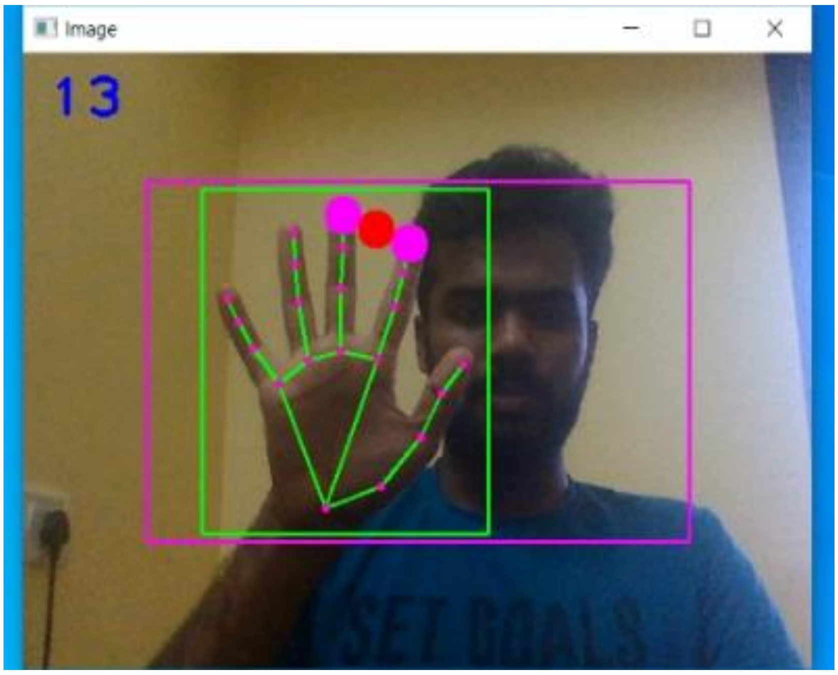
Chapter 6
OUTPUTS

6.1 Hand Detection



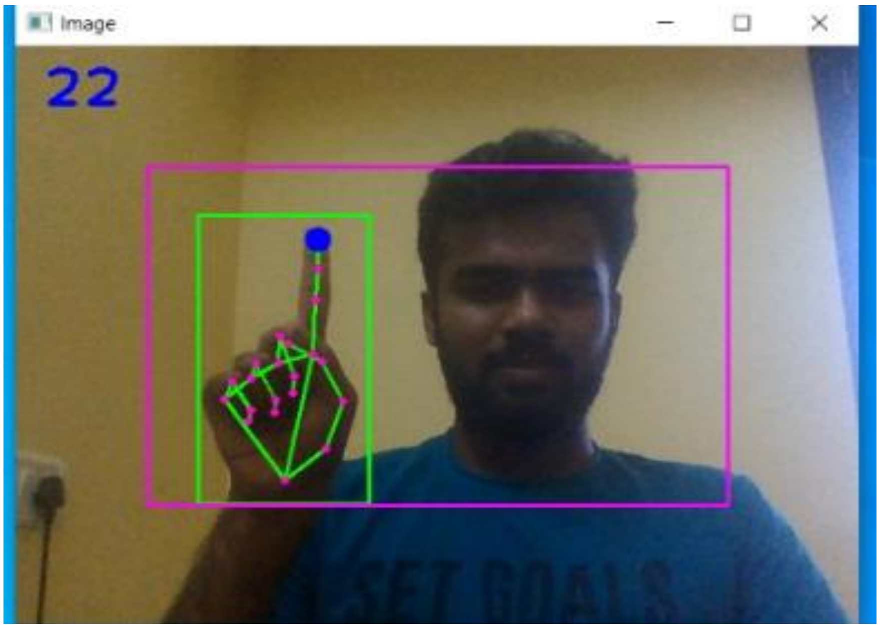
6.2 Left Click

When the Index figure Tip and Middle Figure Top come closure and if the pixel length is lesser than 30, it is configured to do a left click



6.3 Scrolling

When the index figure tip is moved with some pixel then it is configured as scroll.

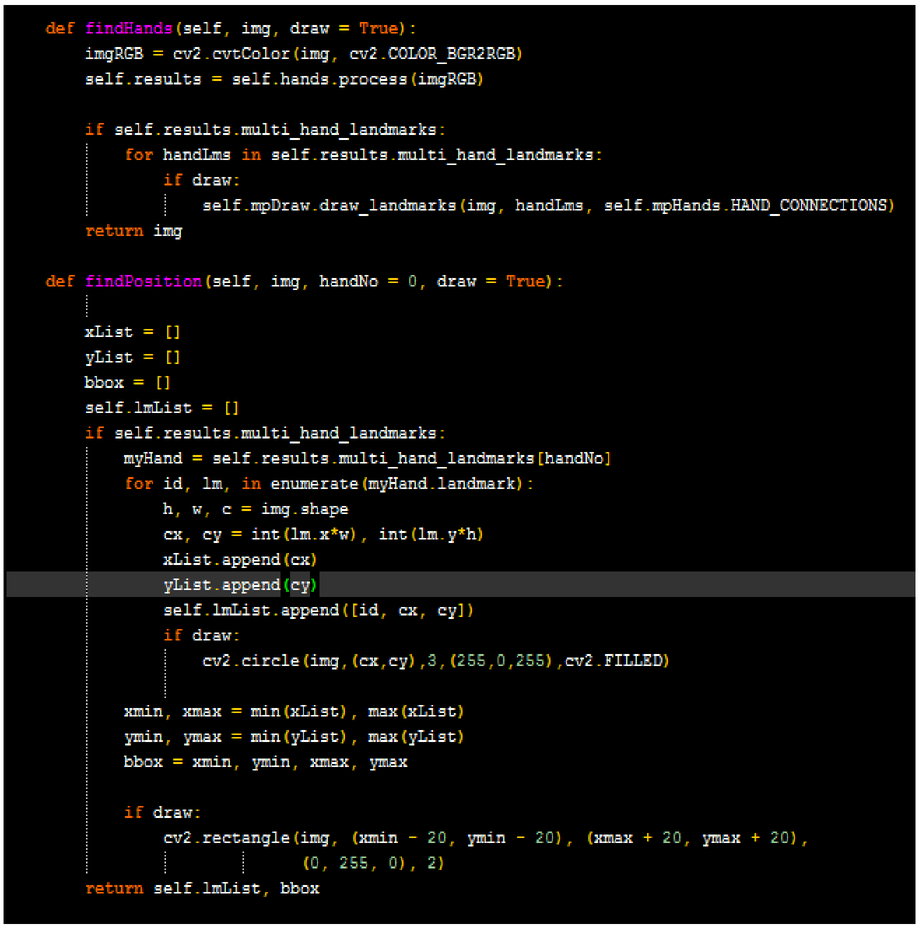


Chapter 7
TESTING:

7.1 White Box Testing

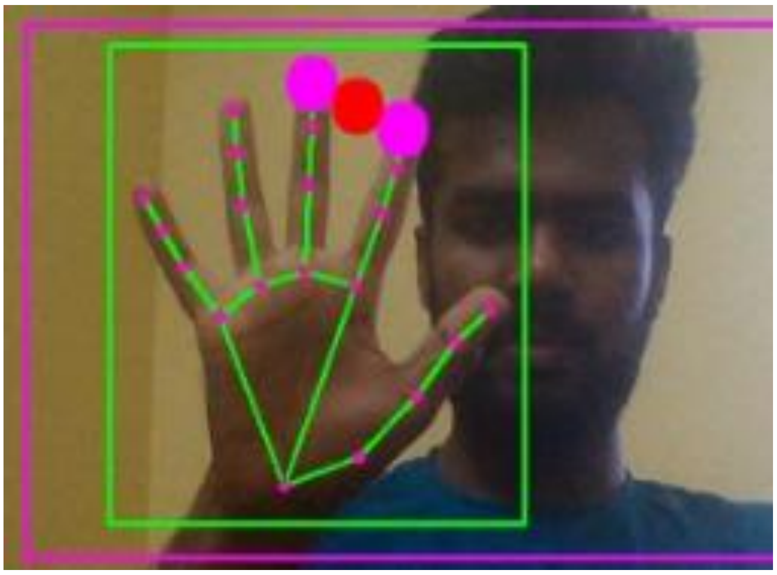
White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability.

As webcam continuously streams the input, handDetector() will run in the infinite loop and when the user provide the input(any handgesture) then the findHands() will detect the hands and provide landmarks of the hand and checks there are multiple hands and return the landmarks accordingly, and the findPosition() should create a list of landmarks(lmList) with min, max of X and Y-axis positions. Ensure that these methods are working correctly by giving some random gestures of the hands.
Also the constructor of the handDetector() can be configured, the parameters maxHands will provide way to detect both the hands or single hand. Run the code by setting the value 1 and 2 and ensure the findPosition() is working properly for the both cases. In addition to it detectionCon will provide way to configure the percentage of the hand detection should consider. Currently it is set to 0.5 means that hand detect should work when the input frame detect atleast 50% of the hand. Change the detectionCon with different percentage and ensure that the algorithms is working without throwing any exceptions.

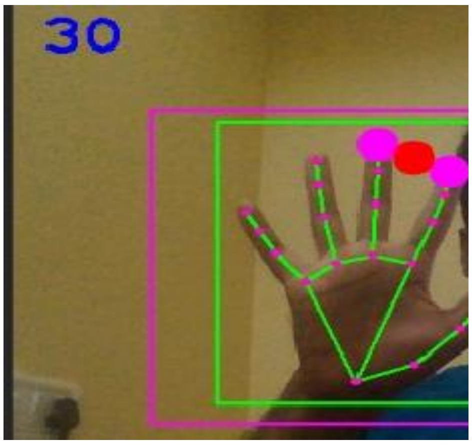


```
def findHands(self, img, draw = True):
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    self.results = self.hands.process(imgRGB)

    if self.results.multi_hand_landmarks:
        for handLms in self.results.multi_hand_landmarks:
            if draw:
                self.mpDraw.draw_landmarks(img, handLms, self.mpHands.HAND_CONNECTIONS)
    return img

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

    xList = []
    yList = []
    bbox = []
    self.lmList = []
    if self.results.multi_hand_landmarks:
        myHand = self.results.multi_hand_landmarks[handNo]
        for id, lm, in enumerate(myHand.landmark):
            h, w, c = img.shape
            cx, cy = int(lm.x*w), int(lm.y*h)
            xList.append(cx)
            yList.append(cy)
            self.lmList.append([id, cx, cy])
            if draw:
                cv2.circle(img,(cx,cy),3,(255,0,255),cv2.FILLED)

        xmin, xmax = min(xList), max(xList)
        ymin, ymax = min(yList), max(yList)
        bbox = xmin, ymin, xmax, ymax

        if draw:
            cv2.rectangle(img, (xmin - 20, ymin - 20), (xmax + 20, ymax + 20),
                          (0, 255, 0), 2)
    return self.lmList, bbox
```

The result of the method should be similar to the figure below.



findDistance() method will provide the pixel distance between finger tips that are identified and highlight the tips with the color using the method cv2.circle when the draw parameter is set to True. Change the different color coding and make sure the tips are highlighted properly. When the draw parameter is set to False test finger tips are not highlighted.
The main result from findDistance() is length that is calculated by taking the pixel distance between the finger tips and it should also be appeared at the top of the image that is rendered.



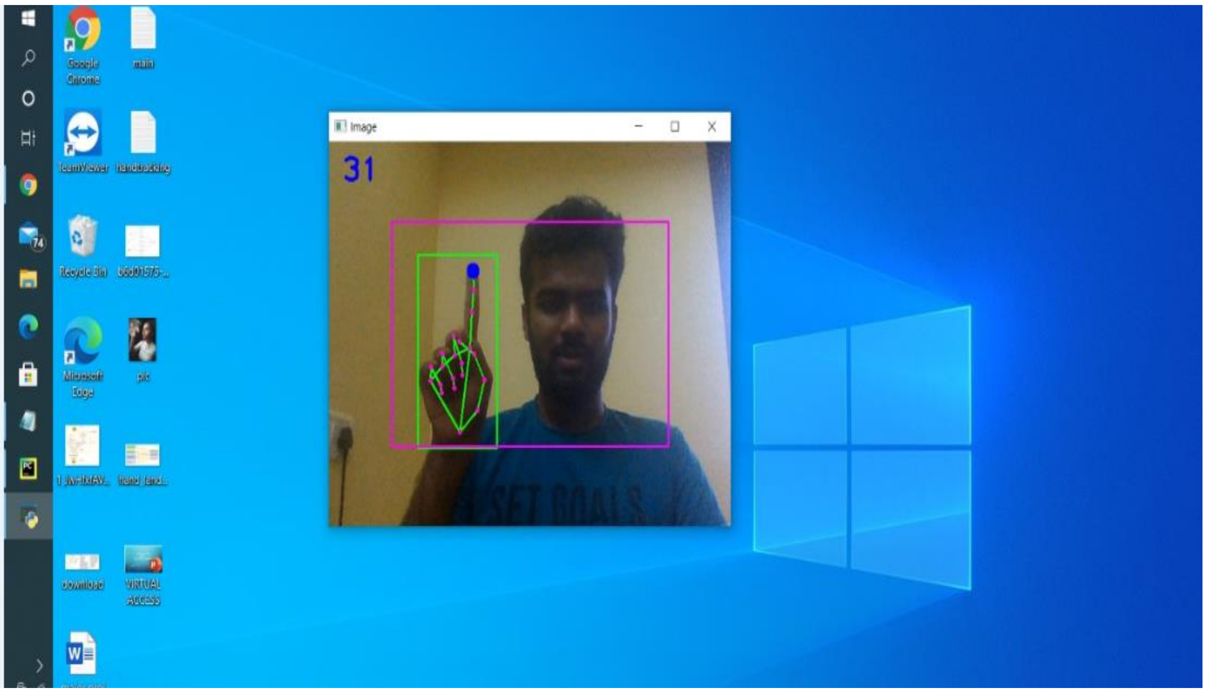
7.2 Black Box Testing

BlackBox is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications.

As the Algorithms detects the hand gestures, Run the algorithm and when the algorithm is running provide some random gestures without freezing the gesture > 1s and check that algorithm is not crashing and provide the results properly.

Provide the inputs that are configured to the algorithm and ensure that hand movement is working properly and the hand detection worked properly then using the hand gestures perform the action of opening an application. (Scrolling the cursor Clicking any application) make sure that the selected application is opened up.

The figure below provides the hand gesture to the move the cursor position to some distance, the action is called Scrolling. In order to perform the mouse click move the index-finger-tip and middle- finger-tip to the pixel distance lesser then 40 and make sure the algorithm has performed the click functionality.



Chapter 8
CONCLUSION AND FUTURE WORKS

In this modern world, where technologies is at the peak, there are many facilities available for offering input to any applications running on the computer systems, some of the inputs can be offered using physical touch and some of them without using physical touch (like speech, hand gestures, head gestures etc.). Using hand gestures many users can handle applications from distance without even touching it. But there are many applications which cannot be controlled using hand gestures as an input. This technique can be very helpful for physically challenged people because they can define the gesture according to their need. The present system which we have implemented although seems to be user friendly as compared to modern device or command based system but it is less robust in detection and recognition as we have seen in the previous step. We need to improve our system and try to build more robust algorithm for both recognition and detection background to work in a normal lighting condition. We also need to extend the system for some more class of gestures as we have implemented it for only 6 classes. However we can use this system to control applications like power point presentation, games, media player, windows picture manager etc.

A new method for hand gesture recognition is introduced in this paper. The hand region is detected from the background by the background subtraction method. Then, the palm and fingers are segmented. On the basis of the segmentation, the fingers in the hand image are discovered and recognized.

The performance of the proposed method highly depends on the result of hand detection.Slight hand movements could affect gesture recognition. Nevertheless, if the hand is steady enough for long enough, the program outputs the correct command.

Chapter 9
BIBILOGRAPY:

[1] S. S. Rautaray and A. Agrawal, Vision Based Hand Gesture Recognition for Human Computer Interaction: A survey, Springer Transaction on Artificial Intelligence Review, pp. 1–54, (2012).
[2] P. Payeur, C. Pasca, A. Cretu and E. M. Petriu, Intelligent Haptic Sensor System for Robotic Manipulation, IEEE Transaction on Instrumentation and Measurement, vol. 54(4), pp. 1583–1592, (2005).
[3] S. Meena, A Study on Hand Gesture Recognition Technique, Master Thesis, Department of Electronics and Communication Engineering, National Institute of Technology, India, (2011).
[4] M. M. Hasan and P. K. Mishra, Hand Gesture Modeling and Recognition using Geometric Features: A Review, Canadian Journal on Image Processing and Computer Vision, vol. 3(1), pp. 12–26, (2012).

[5] B. A. Myers, A Brief History of Human Computer Interaction Technology, ACM Interactions, vol. 5(2), pp. 44–54, (1998).
[6] A. Malima, E. Ozg ¨ ¨ ur and M. C¸ etin, A Fast Algorithm for Vision-Based Hand Gesture Recognition For Robot Control, IEEE Signal Processing and Communications Applications, pp. 1–4, (2006).
[7] Z. Xu, et al., Hand Gesture Recognition and Virtual Game Control Based on 3D Accelerometer and EMG Sensors, In Proceedings of IUI'09, pp. 401–406, (2009).

[8] S. Mitra and T. Acharya, Gesture Recognition: A Survey, IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews, vol. 37(3), pp. 311–324, (2007).
[9] N. X. Tran, Wireless Data Glove for Gesture-Based Robotic Control, 13th International Conference on HCI, vol. 5611, pp. 271–280, (2009).

[10] J. M. Rehg and T. Kanade, Visual Tracking of High DOF Articulated Structures: An Application to Human Hand Tracking, 3rd European Conference on Computer Vision, pp. 35–46, (1994).
[11] Murthy and Jadon, Hand Gesture Recognition using Neural Networks, In 2nd IEEE International Advance Computing Conference (IACC), pp. 134–138, (2010).
[12] S. K. Kang, M. Y. Nam and P. K. Rhee, Color Based Hand and Finger Detection Technology for user Interaction, IEEE International Conference on Convergence and Hybrid Information Technology, pp. 229–236, (2008).

[13] N. H. Dardas and N. D. Georganas, Real-Time Hand Gesture Detection and Recognition using Bag-of-Features and Support Vector Machine Techniques, IEEE Transaction on Instrumentations and Measurement, vol. 60(11), pp. 3592–3607, November (2011).

[14] S. S. Rautaray and A. Agrawal, A Novel Human Computer Interface Based on Hand Gesture Recognition using Computer Vision Techniques, In Proceedings of ACM IITM'10, pp. 292–296, (2010).

[15] N. A. Ibraheem, R. Z. Khan and M. M. Hasan, Comparative Study of Skin Color Based Segmentation Techniques, International Journal of Applied Information Systems (IJAIS), vol. 5(10), pp. 24–38, (2013).