GESTURE CONTROLLED VIRTUAL MOUSE

A Major Project Report Submitted In partial fulfillment of the requirements or the award of the degree of

Bachelor of Technology in
Computer Science and Engineering

by

R. Sai Kiran	**18N31A05J4**
S. Mohith	**19N35A0523**
Sai Abhinav V	**18N31A05J7**

Under the esteemed guidance of

Mr. M. Sandeep
Associate Professor

Department of Computer Science and Engineering
Malla Reddy College of Engineering & Technology
(Autonomous Institution- UGC, Govt. of India)
(Affiliated to JNTUH, Hyderabad, Approved by AICTE, NBA &NAAC with 'A' Grade)
Maisammaguda, Kompally, Dhulapally, Secunderabad – 500100 website: www.mrcet.ac.in

2018-2022

Malla Reddy College of Engineering & Technology

(Autonomous Institution- UGC, Govt. of India)
(Affiliated to JNTUH, Hyderabad, Approved by AICTE, NBA &NAAC with 'A' Grade)
Maisammaguda, Kompally, Dhulapally, Secunderabad – 500100 website: www.mrcet.ac.in

CERTIFICATE

This is to certify that this is the bonafide record of the project entitled "**Gesture Controlled Virtual Mouse**", submitted by R. Sai Kiran (18N31A05J4), S. Mohith (19N35A0523) and Sai Abhinav V (18N31A05J7) of B.Tech in the partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science and Engineering, Department of CSE during the year 2021-2022. The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

Internal Guide

Mr. M. Sandeep
Associate Professor

Head of the Department

Dr. S. Shanthi
Professor

External Examiner

DECLARATION

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

Name of Student(s)	:	R. Sai Kiran
		S. Mohith
		Sai Abhinav V
Hall Ticket Number(s):		18N31A05J4
		19N35A0523
		18N31A05J7
Degree	:	Bachelor of Technology in Computer Science and Engineering
Department	:	Computer Science and Engineering
Title of the project	:	Gesture Controlled Virtual Mouse

ACKNOWLEDGEMENT

We feel ourselves honored and privileged to place our warm salutation to our college Malla Reddy College of Engineering and Technology (UGC-Autonomous) and our Director **Dr. V. S. K Reddy** and our **Principal Dr. S. Srinivasa Rao**, who gave us the opportunity to have experience in engineering and profound technical knowledge.

We express our heartiest thanks to our Head of the Department **Dr. S. Shanthi** for encouraging us in every aspect of our project and helping us realize our full potential.

We would like to thank our internal guide and Project Coordinator **Mr. M. Sandeep** for his regular guidance and constant encouragement. We are extremely grateful to his valuable suggestions and unflinching co-operation throughout project work.

We would like to thank our class in charge **Mr. M. SambaSivudu** who in spite of being busy with his duties took time to guide and keep us on the correct path.

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

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

with regards and gratitude

R. Sai Kiran - 18N31A05J4

S. Mohith - 19N35A0523

Sai Abhinav V - 18N31A05J7

ABSTRACT

Communication plays a crucial part in human life. It encourages a man to pass on his sentiments, feelings and messages by talking, composing or by utilizing some other medium. Gesture based communication is the main method for Communication for the discourse and hearing weakened individuals. There are two sorts of sign recognition methods: image- based and sensor-based strategies. Image based approach is utilized as a part of this project that manages communication via gestures motions to distinguish and track the signs and change over them into the relating discourse and content. Gesture Controlled Virtual Mouse makes human computer interaction simple by making use of Hand Gesture. The computer requires almost no direct contact. All I/O operations can be virtually controlled by using static and dynamic hand gestures. This project makes use of the state-of-art Machine Learning and Computer Vision algorithms to recognize hand gestures, which works smoothly without any additional hardware requirements. It leverages models such as CNN implemented by Media Pipe running on top of pybind11. The Virtual Mouse provides an infrastructure between the user and the machine. It enables the user to interact with a machine without the need for any mechanical or physical devices, and even allows to control mouse functions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF OUTPUTS

1. INTRODUCTION

This chapter gives an overview about the purpose, aim, objectives, background and operation environment of the system.

1.1 PURPOSE, AIM AND OBJECTIVES:

The aim of this project is to develop a Virtual Mouse application that targets a few aspects of significant development. For starters, this project aims to eliminate the needs of having a physical mouse while able to interact with the computer system through webcam by using various image processing techniques. Other than that, this project aims to develop a Virtual Mouse application that can be operational on all kind of surfaces and environment.

The goal is to manage computers and other devices with gestures rather than pointing and clicking a mouse or touching a display directly. Backers believe that the approach can make it not only easier to carry out many existing chores but also take on trickier tasks such as creating 3-D models, browsing medical imagery during surgery without touching anything. Also, to reduce the cost of having hardware.

1.2 BACKGROUND OF PROJECT:

A mouse, in computing terms is a pointing device that detects two-dimensional movements relative to a surface. This movement is converted into the movement of a pointer on a display that allows to control the Graphical User Interface (GUI) on a computer platform. There are a lot of different types of mouses that have already existed in the modern day's technology, there's the mechanical mouse that determines the movements by a hard rubber ball that rolls around as the mouse is moved. Years later, the optical mouse was introduced that replace the hard rubber ball to a LED sensor to detects table top movement and then sends off the information to the computer for processing. On the year 2004, the laser mouse was then introduced to improve the accuracy movement with the

slightest hand movement, it overcome the limitations of the optical mouse which is the difficulties to track high-gloss surfaces.

However, no matter how accurate can it be, there are still limitations exist within the mouse itself in both physical and technical terms. For example, a computer mouse is a consumable hardware device as it requires replacement in the long run, either the mouse buttons were degraded that causes inappropriate clicks, or the whole mouse was no longer detected by the computer itself. Despite the limitations, the computer technology still continues to grow, so does the importance of the human computer interactions.

Ever since the introduction of a mobile device that can be interact with touch screen technology, the world is starting to demand the same technology to be applied on every technological device, this includes the desktop system. However, even though the touch screen technology for the desktop system is already exist, the price can be very steep. Therefore, a virtual human computer interaction device that replaces the physical mouse or keyboard by using a webcam or any other image capturing devices can be an alternative way for the touch screen. This device which is the webcam will be constantly utilized by a software that monitors the gestures given by the user in order to process it and translate to motion of a pointes, as similar to a physical mouse.

1.3 SCOPE OF PROJECT:

Virtual Mouse that will soon to be introduced to replace the physical computer mouse to promote convenience while still able to accurately interact and control the computer system. To do that, the software requires to be fast enough to capture and process every image, in order to successfully track the user's gesture. Therefore, this project will develop a software application with the aid of the latest software coding technique and the open-source computer vision library also known as the OpenCV.

The scope of the project is as below:

- Real time application.
- User friendly application.
- Removes the requirement of having a physical mouse

1.4 MODULES DESCRIPTION:

This project is composed of four main modules which also include sub modules:

1. **Gesture Controller:** This is the main module of this project. Its main task is to classify the hands whether left hand or right hand from the webcam and then start the hard recognition process.
2. **Hand Recognition:** This module deals with recognizing the state of hand. It performs the operations like first finding the palm and then finding fingers. Next it calculates the angle and distance between fingers so that gestures can be recognized.
3. **Image Processing:** This is an independent module that takes all the images from the video as input and performs various image classifications such as calibrating the image, get the grey scale values, finding only region of interest from the image etc.
4. **Controller:** Based on the results from all the above modules this module finally recognises the gestures and performs respective events on the monitor screen. Different operations such as scrolling vertically or horizontally, changing brightness, drag and drop items, left click and right click, double click etc.

2. SYSTEM ANALYSIS

In this chapter, we will discuss and analyze about the developing process of Audit Control including software requirement specification (SRS) and comparison between existing and proposed system. The functional and non-functional requirements are included in SRS part to provide complete description and overview of system requirement before the developing process is carried out. Besides that, existing vs. proposed provides a view of how the proposed system will be more efficient than the existing one.

2.1 HARDWARE AND SOFTWARE REQUIREMENTS

2.1.1 HARDWARE REQUIREMENTS:

Processor	**:**	Intel or AMD.
Ram	**:**	2 GB or above.
Hard disk	**:**	40GB or above.
Peripheral device	**:**	webcam with at least 30 frames per second, 640 x 480 resolution

2.1.2 SOFTWARE REQUIREMENTS:

Technology/Language	**:**	Python
IDE	:	PyCharm, VScode, Sublime
Operating System	:	Windows, Linux, Mac.
Python version	:	3.6x and later versions
Library	:	Open CV

***Pre required/ installed web cam drivers**.

2.2 SOFTWARE REQUIREMENT SPECIFICATION:

2.2.1 SRS:

Software Requirement Specification (SRS) is the starting point of the software developing activity. As system grew more complex it became evident that the goal of the entire system cannot be easily comprehended. Hence the need for the requirement phase arose. The software project is initiated by the client needs. The SRS is the means of translating the ideas of the minds of clients (the input) into a formal document (the output of the requirement phase.)

The SRS phase consists of two basic activities:

1) **Problem/Requirement Analysis:**

The process is order and more nebulous of the two, deals with understand the problem, the goal and constraints.

2) **Requirement Specification:**

Here, the focus is on specifying what has been found giving analysis such as representation, specification languages and tools, and checking the specifications are addressed during this activity.

The Requirement phase terminates with the production of the validate SRS document. Producing the SRS document is the basic goal of this phase.

2.2.2 ROLE OF SRS:

The purpose of the Software Requirement Specification is to reduce the communication gap between the clients and the developers. Software Requirement Specification is the medium through which the client and user needs are accurately specified. It forms the basis of software development. A good SRS should satisfy all the parties involved in the system.

2.2.3 SCOPE:

This document is the only one that describes the requirements of the system. It is meant for the use by the developers, and will also be the basis for validating the final delivered system. Any changes made to the requirements in the future will have to go through a formal change approval process. The developer is responsible for asking for clarifications, where necessary, and will not make any alterations without the permission of the client.

2.2.4 EXISTING SYSTEM:

The current system is comprised of a generic mouse and trackpad monitor control system, as well as the absence of a hand gesture control system. The use of a hand gesture to access the monitor screen from a distance is not possible. Even though it is primarily attempting to implement, the scope is simply limited in the virtual mouse field. The existing virtual mouse control system consists of simple mouse operations using a hand recognition system, in which we can control the mouse pointer, left click, right click, and drag, and so on. The use of hand recognition in the future will not be used. Even though there are a variety of systems for hand recognition, the system they used is static hand recognition, which is simply a recognition of the shape made by the hand and the definition of action for each shape made, which is limited to a few defined actions and causes a lot of confusion. As technology advances, there are more and more alternatives to using a mouse.

2.2.4.1 DRAWBACKS OF EXISTING SYSTEM:

- Always a need of operating a physical device.
- Not convenient and not a cost-effective solution.

2.2.5 PROPOSED SYSTEM:

Using our project we could make use of the laptop's web-cam and by recognizing the hand gesture we could control mouse and perform basic operations like mouse pointer controlling, select and deselect using left click, and some additional features like scrolling, drag and drop, multiple item selection, volume control etc.

The project done is a "Zero Cost" hand recognition system for laptops, which uses simple algorithms to determine the hand, hand movements and by assigning an action for each movement. The system we are implementing which is been written in python code be much more responsive and is easily implemented since python is a simple language and is platform independent with flexibility and is also portable.

2.2.5.1 ADVANTAGES OF PROPOSED SYSTEM:

- Any new product should either make human life more comfortable, more productive or more fun.
- Provides greater flexibility than the existing system.
- Easy to modify and adapt.
- Less prone to physical damage due to absence of a fixed physical device.
- Avoid the mouse-related wrist damage like CTS & RSI.
- Also, there is a certain degree of fun & entertainment associated with the whole idea.

3. TECHNOLOGIES USED

3.1 COMPUTER VISION:

3.1.1 What is computer vision?

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs — and take actions or make recommendations based on that information. If AI enables computers to think, computer vision enables them to see, observe and understand.

Computer vision trains machines to perform these functions, but it has to do it in much less time with cameras, data and algorithms rather than retinas, optic nerves and a visual cortex. Because a system trained to inspect products or watch a production asset can analyze thousands of products or processes a minute, noticing imperceptible defects or issues, it can quickly surpass human capabilities.

3.1.2 How does computer vision work?

Computer vision needs lots of data. It runs analyses of data over and over until it discerns distinctions and ultimately recognize images. For example, to train a computer to recognize automobile tires, it needs to be fed vast quantities of tire images and tire-related items to learn the differences and recognize a tire, especially one with no defects.

Two essential technologies are used to accomplish this: a type of machine learning called deep learning and a convolutional neural network (CNN).

Machine learning uses algorithmic models that enable a computer to teach itself about the context of visual data. If enough data is fed through the model, the computer will “look” at the data and teach itself to tell one image from another. Algorithms enable the machine to learn by itself, rather than someone programming it to recognize an image.

A CNN helps a machine learning or deep learning model “look” by breaking images down into pixels that are given tags or labels. It uses the labels to perform convolutions (a mathematical operation on two functions to produce a third function) and makes predictions about what it is “seeing.” The neural network runs convolutions and checks the accuracy of

its predictions in a series of iterations until the predictions start to come true. It is then recognizing or seeing images in a way similar to humans.

Much like a human making out an image at a distance, a CNN first discerns hard edges and simple shapes, then fills in information as it runs iterations of its predictions. A CNN is used to understand single images. A recurrent neural network (RNN) is used in a similar way for video applications to help computers understand how pictures in a series of frames are related to one another.

IMPLEMENTATION OF COMPUTER VISION (CV) / PYTHON OPEN CV MODULE

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. cv2.imread() method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix.

There is a package cv2 which is already inbuilt in python. So to import the package we use,

```
>>> import cv2
```

3.1.3 COMPUTER VISION / OPEN CV EXAMPLES

Here are a few examples of established computer vision tasks:

• **Image classification** sees an image and can classify it (a dog, an apple, a person's face). More precisely, it is able to accurately predict that a given image belongs to a certain class. For example, a social media company might want to use it to automatically identify and segregate objectionable images uploaded by users.

• **Object detection** can use image classification to identify a certain class of image and then detect and tabulate their appearance in an image or video. Examples include detecting damages on an assembly line or identifying machinery that requires maintenance.

• **Object tracking** follows or tracks an object once it is detected. This task is often executed with images captured in sequence or real-time video feeds. Autonomous vehicles, for example, need to not only classify and detect objects such as pedestrians, other cars and road infrastructure, they need to track them in motion to avoid collisions and obey traffic laws.

• **Content-based image retrieval** uses computer vision to browse, search and retrieve images from large data stores, based on the content of the images rather than metadata tags

associated with them. This task can incorporate automatic image annotation that replaces manual image tagging.

3.2 HAND TRACKING / MEDIAPIPES:

Mediapipe is a framework mainly used for building audio, video, or any time series data. With the help of the MediaPipe framework, we can build very impressive pipelines for different media processing functions.

Some of the major applications of MediaPipe.

- Multi-hand Tracking
- Face Detection
- Object Detection and Tracking
- Objectron: 3D Object Detection and Tracking
- AutoFlip: Automatic video cropping pipeline etc.

In this project mediapipe is used for face detection and multi hand tracking purpose.

IMPLEMENTATION OF MEDIAPIPE

Before the implementation of mediapipe open cv and mediapipe needs to be installed in the system. To install mediapipe follow this command in the terminal

>>> pip install mediapipe

To implement mediapipe use this command.

>>> import mediapipe as np

WORKING OF HAND DETECTION MECHANISM

The MediaPipe pipeline utilizes multiple models like, a palm detection model that returns an oriented hand bounding box from the full image. The cropped image region is fed to a hand landmark model defined by the palm detector and returns high-fidelity 3D hand key points.

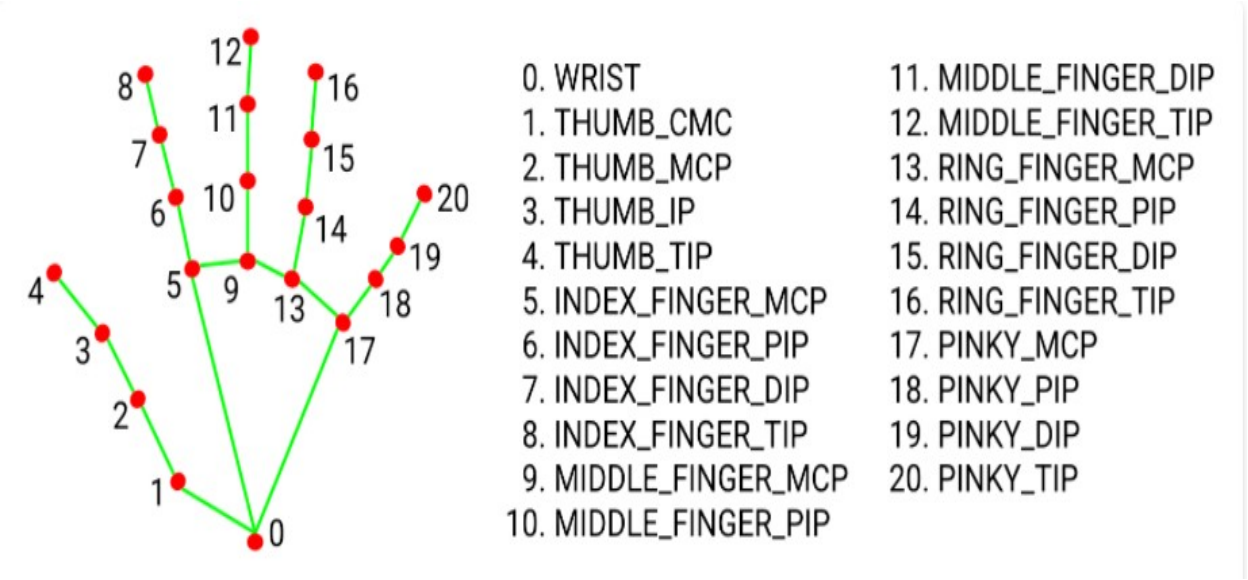

FIGURE 3.2: MEDIAPIPE DIAGRAM

4. SYSTEM DESIGN & UML DIAGRAMS

System design is transition from a user-oriented document to programmers or data base personnel. The design is a solution, how to approach to the creation of a new system. This is composed of several steps. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Designing goes through logical and physical stages of development, logical design reviews the present physical system, prepare input and output specification, details of implementation plan and prepare a logical design walkthrough.

4.1 SOFTWARE DESIGN:

In designing the software following principles are followed:

1) **Modularity and partitioning**: Software is designed such that, each system should consist of hierarchy of modules and serve to partition into separate function.
2) **Coupling:** Modules should have little dependence on other modules of a system.
3) **Cohesion:** Modules should carry out in a single processing function.
4) **Shared use:** Avoid duplication by allowing a single module be called by other that need the function it provides.

4.2 ARCHITECTURE:

Architecture diagram is a diagram of a system, in which the principal parts or functions are represented by blocks connected by lines that show the relationships of the blocks. The block diagram is typically used for a higher level, less detailed description aimed more at understanding the overall concepts and less at understanding the details of implementation.

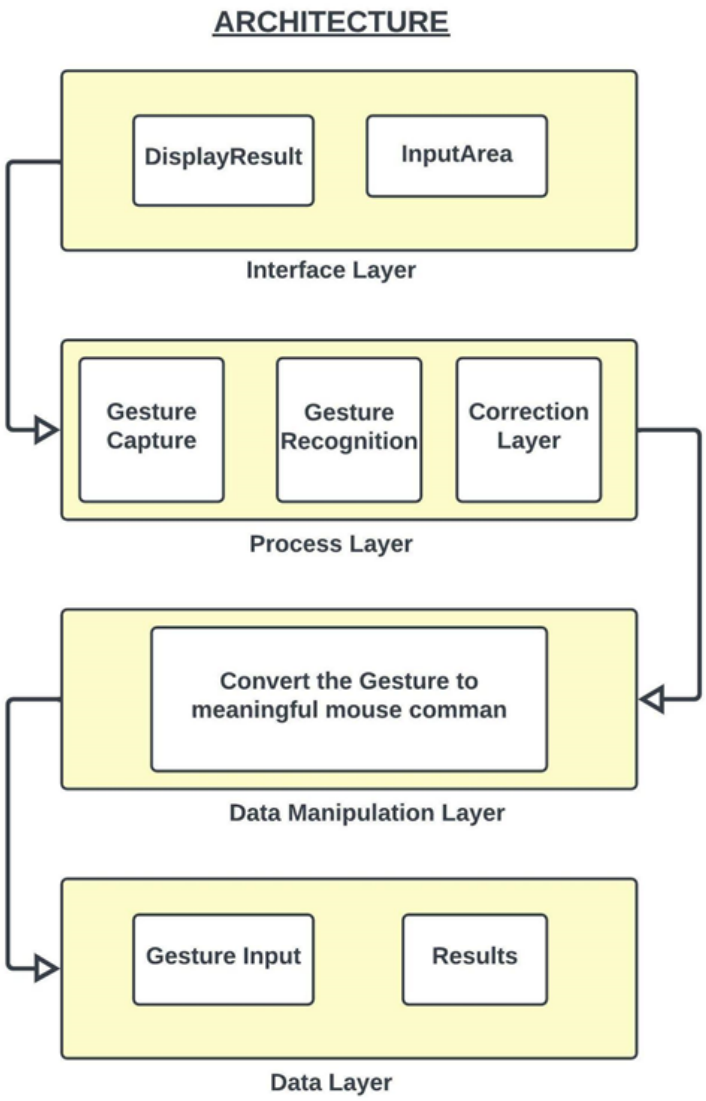

FIGURE 4.2: ARCHITECTURE

4.3 UNIFIED MODELING LANGUAGE (UML):

The unified modeling is a standard language for specifying, visualizing, constructing and documenting the system and its components is a graphical language which provides a vocabulary and set of semantics and rules. The UML focuses on the conceptual and physical representation of the system. It captures the decisions and understandings about systems that must be constructed. It is used to understand, design, configure and control information about the systems.

Depending on the development culture, some of these artifacts are treated more or less formally than others. Such artifacts are not only the deliverables of a project; they are also critical in controlling, measuring, and communicating about a system during its development and after its deployment.

The UML addresses the documentation of a system's architecture and all of its details. The UML also provides a language for expressing requirements and for tests. Finally, the UML provides a language for modeling the activities of project planning and release management.

4.3.1 BUILDING BLOCKS OF UML:

The vocabulary of the UML encompasses three kinds of building blocks:

- ✓ Things.
- ✓ Relationships.
- ✓ Diagrams.

4.3.1.1 Things in the UML:

Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

There are four kinds of things in the UML:

- ✓ Structural things.
- ✓ Behavioral things.
- ✓ Grouping things.
- ✓ Annotational things.

1. **Structural things** are the nouns of UML models. The structural things used in the project design are:
   - ✓ First, a **class** is a description of a set of objects that share the same attributes, operations, relationships and semantics.

Window
origin size

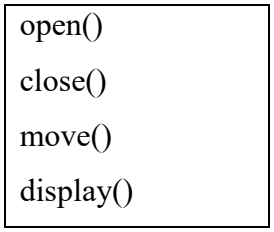

Fig: Classes

- ✓ Second, a **use case** is a description of set of sequence of actions that a system performs that yields an observable result of value to particular actor.

Fig: Use Cases

- ✓ Third, a node is a physical element that exists at runtime and represents a computational resource, generally having at least some memory and often processing capability.

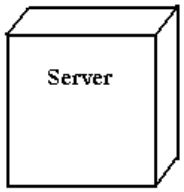

Fig: Nodes

2. **Behavioral things** are the dynamic parts of UML models. The behavioral thing used is:
   - ✓ Interaction: An interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. An interaction involves a number of other elements, including messages, action sequences (the behavior invoked by a message, and links (the connection between objects).

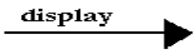

Fig: Messages

4.3.1.2 Relationships in the UML:

There are four kinds of relationships in the UML:

- ✓ Dependency.
- ✓ Association.
- ✓ Generalization.
- ✓ Realization.

✓ A **dependency** is a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing (the dependent thing).

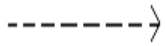

Fig: Dependencies

✓ An **association** is a structural relationship that describes a set links, a link being a connection among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts.

Fig: Association

✓ A **generalization** is a specialization/ generalization relationship in which objects of thespecialized element (the child) are substitutable for objects of the generalized element(the parent).

Fig: Generalization

✓ A **realization** is a semantic relationship between classifiers, where in one classifier specifies a contract that another classifier guarantees to carry out.

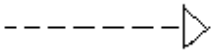

Fig: Realization

4.3.2 UML DIAGRAMS:

4.3.2.1 CLASS DIAGRAM:

A class is a representation of an object and, in many ways; it is simply a template from which objects are created. Classes form the main building blocks of an object-oriented application. Although thousands of students attend the university, you would only model one class, called Student, which would represent the represent the entire collection of students.

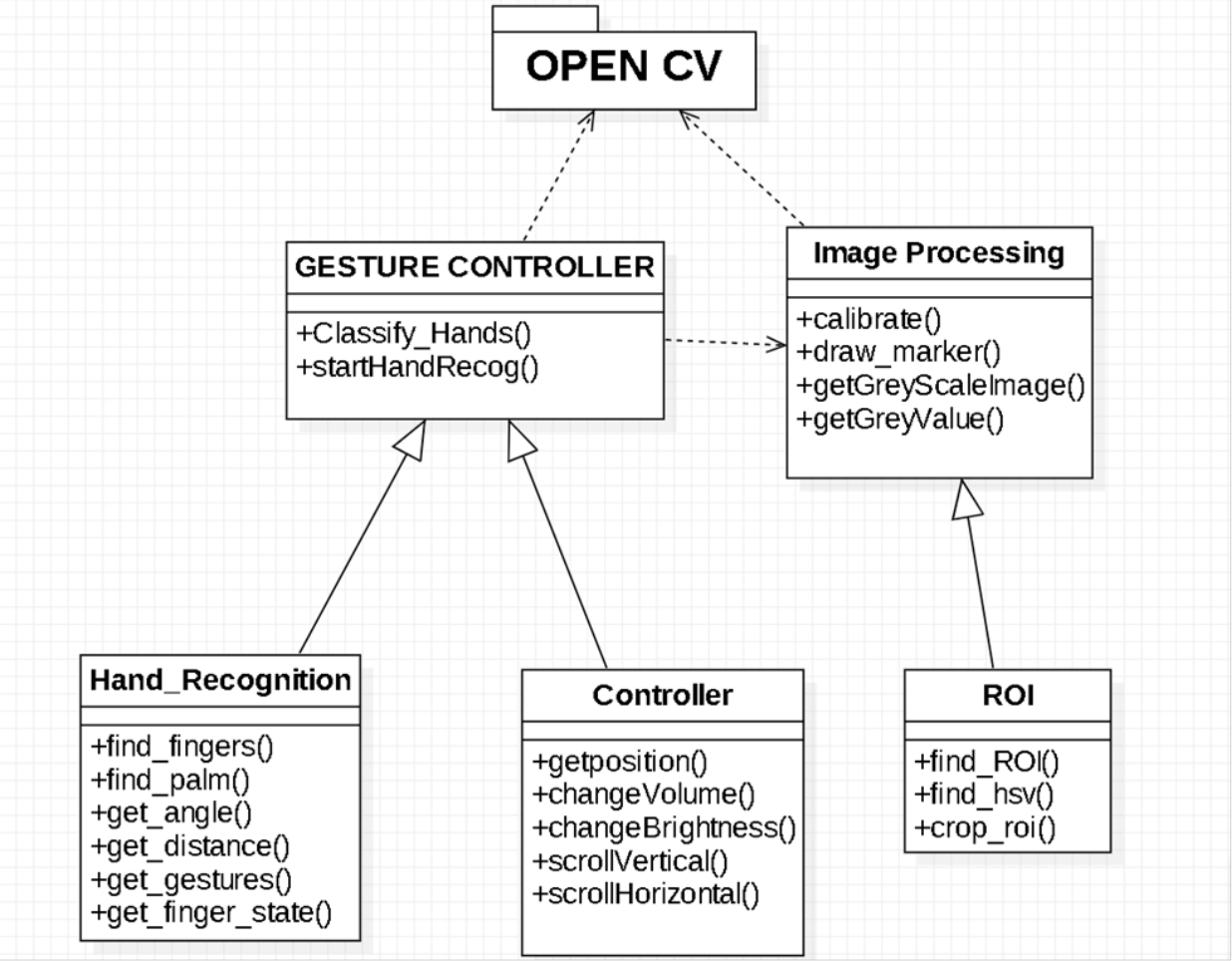

FIGURE 4.3.2.1: CLASS DIAGRAM

4.3.2.2 USE CASE DIAGRAM:

A use case diagram is a graph of actors set of use cases enclosed by a system boundary, communication associations between the actors and users and generalization among use cases. The use case model defines the outside (actors) and inside (use case) of the system's behavior.

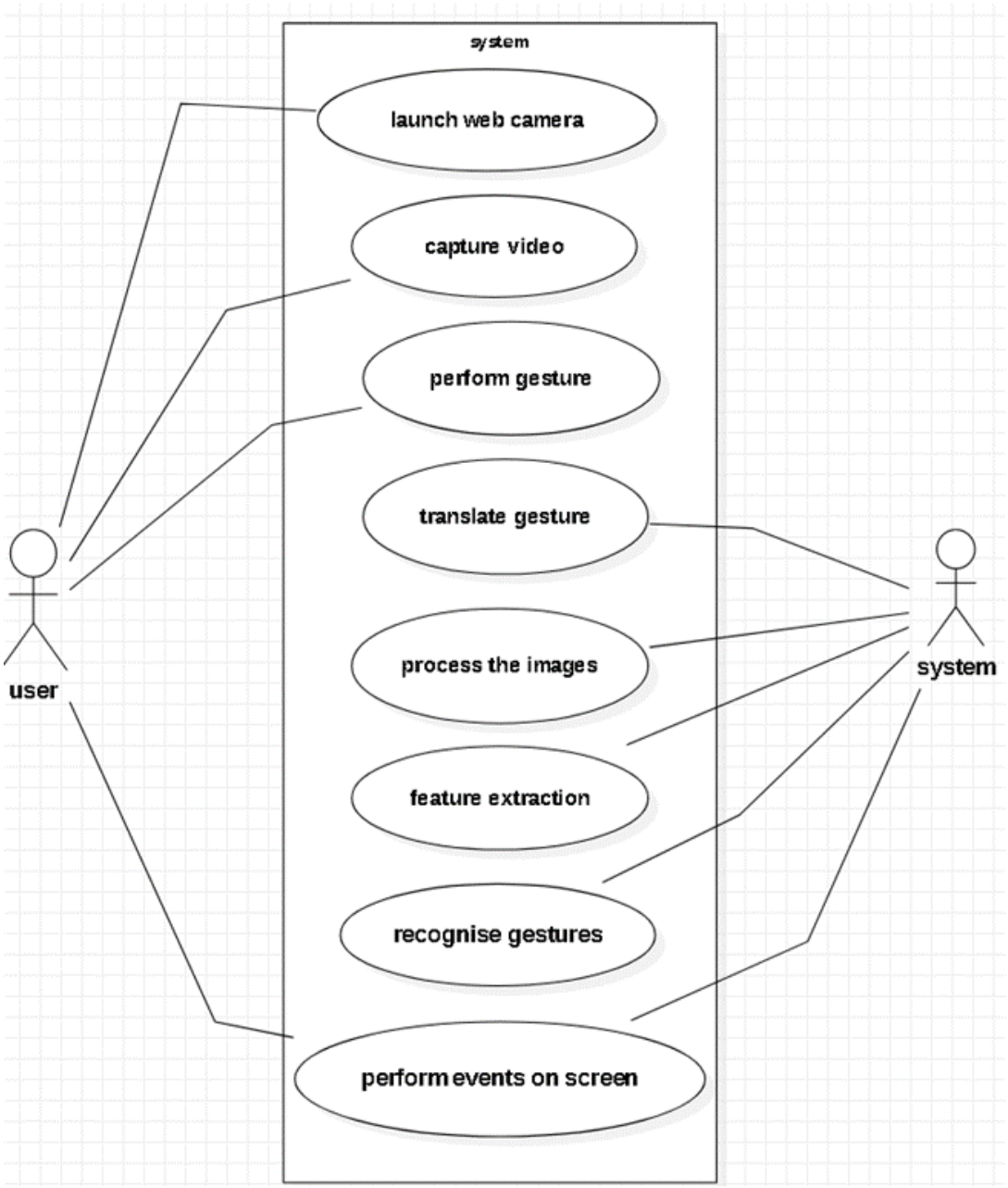

FIGURE 4.3.2.2: USE CASE DIAGRAM

4.3.2.3 SEQUENCE DIAGRAM:

Sequence diagram are used to represent the flow of messages, events and actions between the objects or components of a system. Time is represented in the vertical direction showing the sequence of interactions of the header elements, which are displayed horizontally at the top of the diagram.

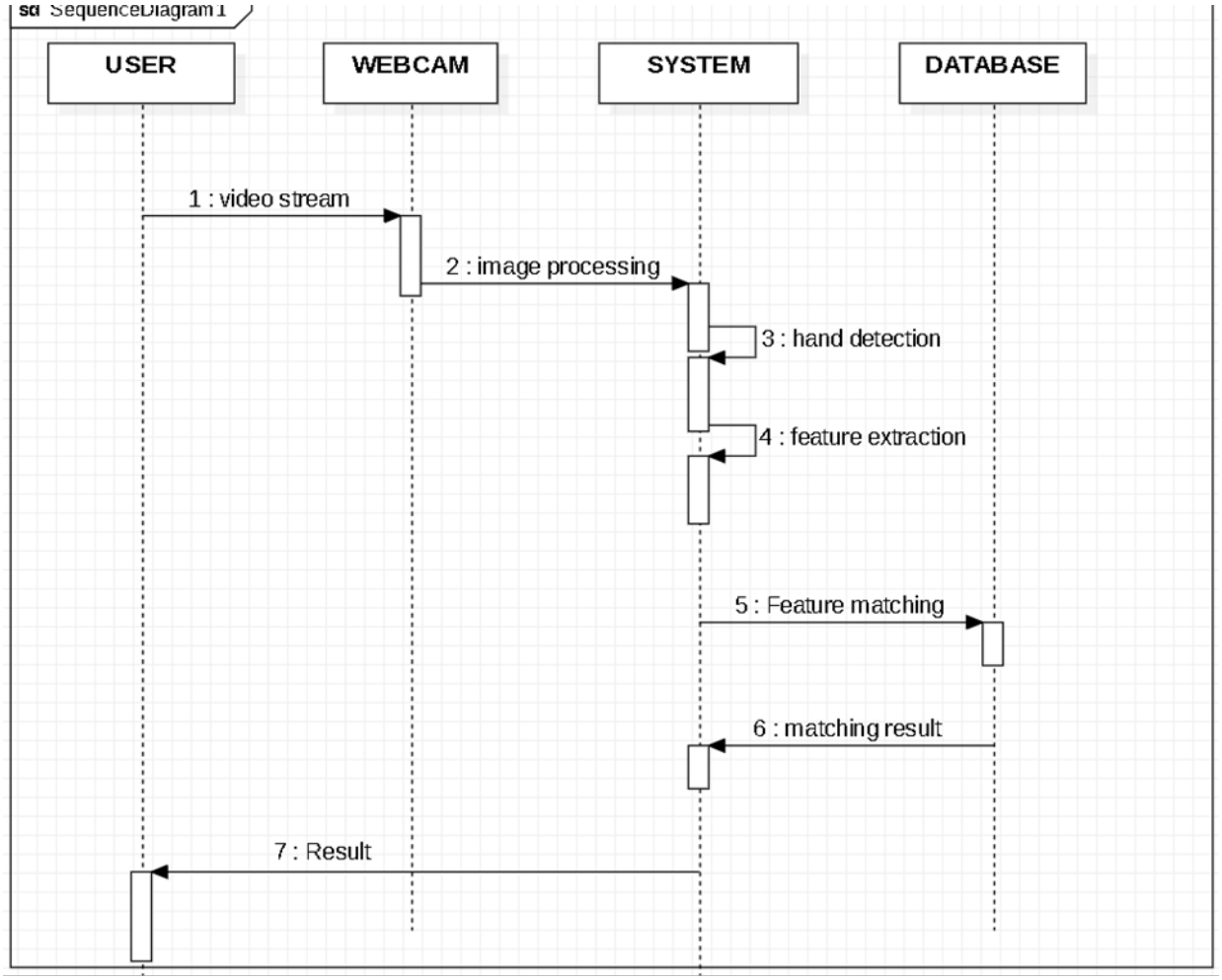

FIGURE 4.3.2.3: SEQUENCE DIAGRAM FOR GPS

4.3.2.4 ACTIVITY DIAGRAM:

Activity diagram represent the business and operational workflows of a system. An Activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state.

So, what is the importance of an Activity diagram, as opposed to a State diagram? A State diagram shows the different states an object is in during the lifecycle of its existence in the system, and the transitions in the states of the objects. These transitions depict the activities causing these transitions, shown by arrows.

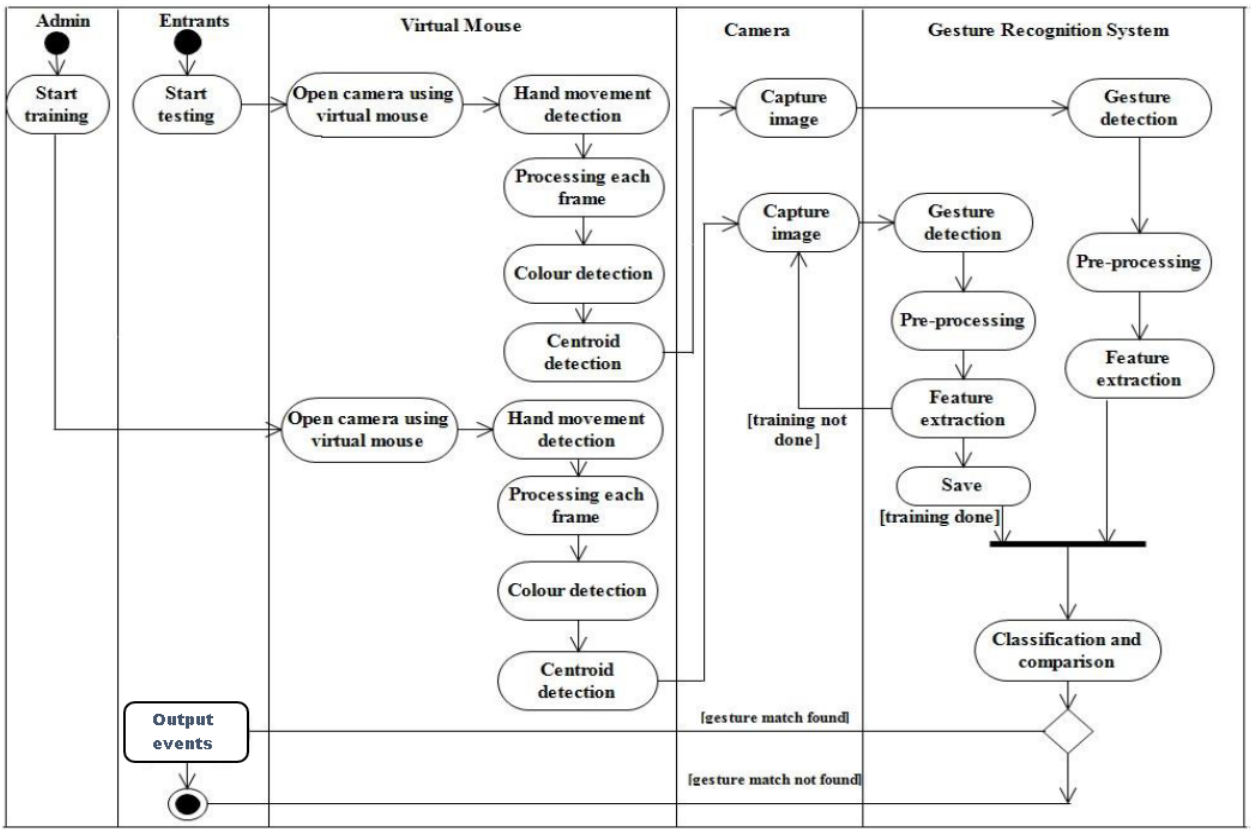

FIGURE 4.3.2.4: ACTIVITY DIAGRAM

4.3.2.5 STATE CHART DIAGRAM:

State chart diagram is used to describe the states of different objects in its life cycle. So the emphasis is given on the state changes upon some internal or external events. These states of objects are important to analyse and implement them accurately. State chart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

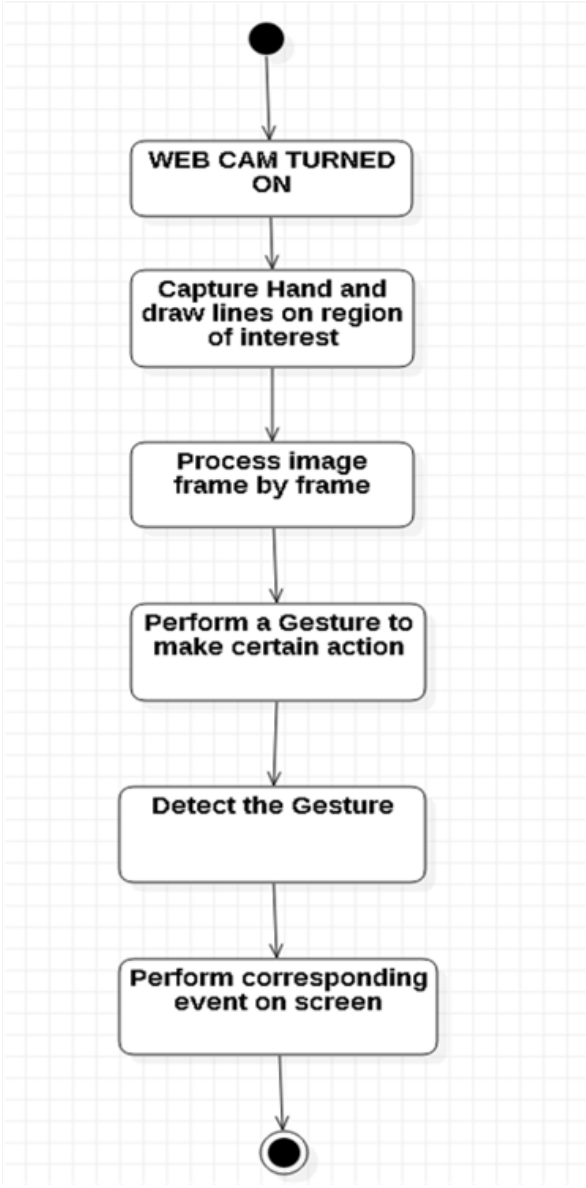

FIGURE 4.3.2.5: STATE CHART

4.3.2.6 DEPLOYMENT DIAGRAM:

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed. So deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

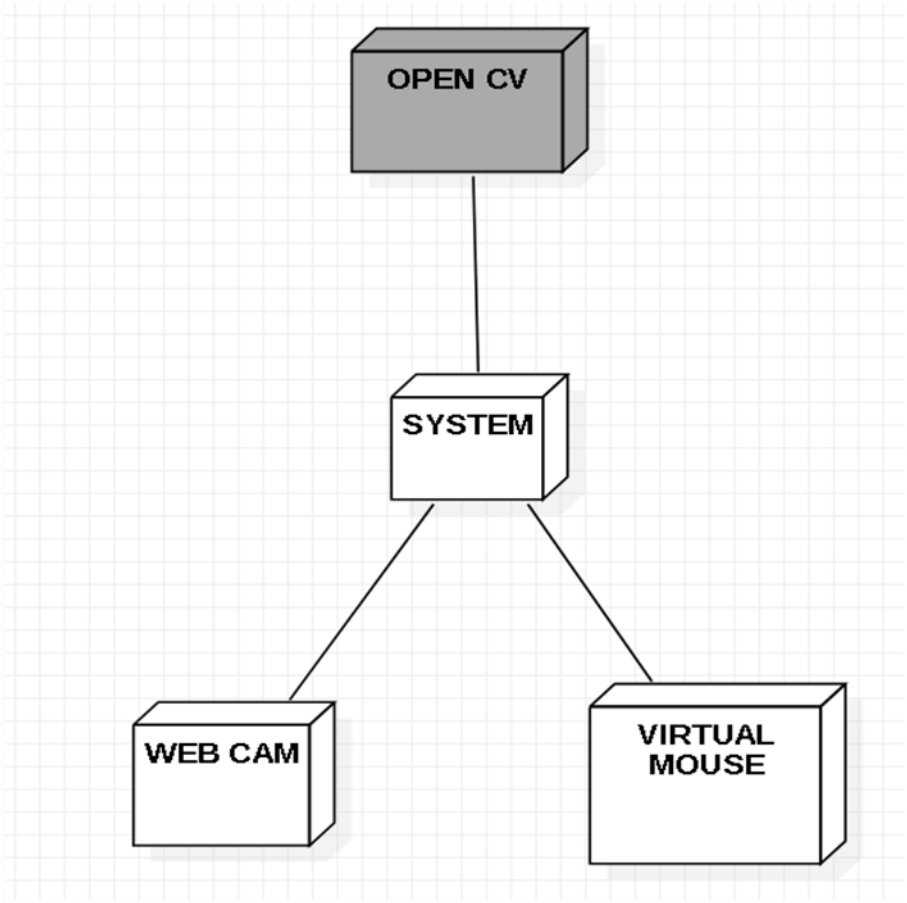

FIGURE 4.3.2.6: DEPLOYMENT DIAGRAM

4.3.2.7 COMPONENT DIAGRAM:

In the component view, the component diagram (CPDs) shows the dependencies between compilation units, typically including sources files, and the runtime components of the final system. In the process view, the CDPs show the dependencies between processes (Schedulable units that may be selected for execution on the host machine's processor) in terms of the runtime components.

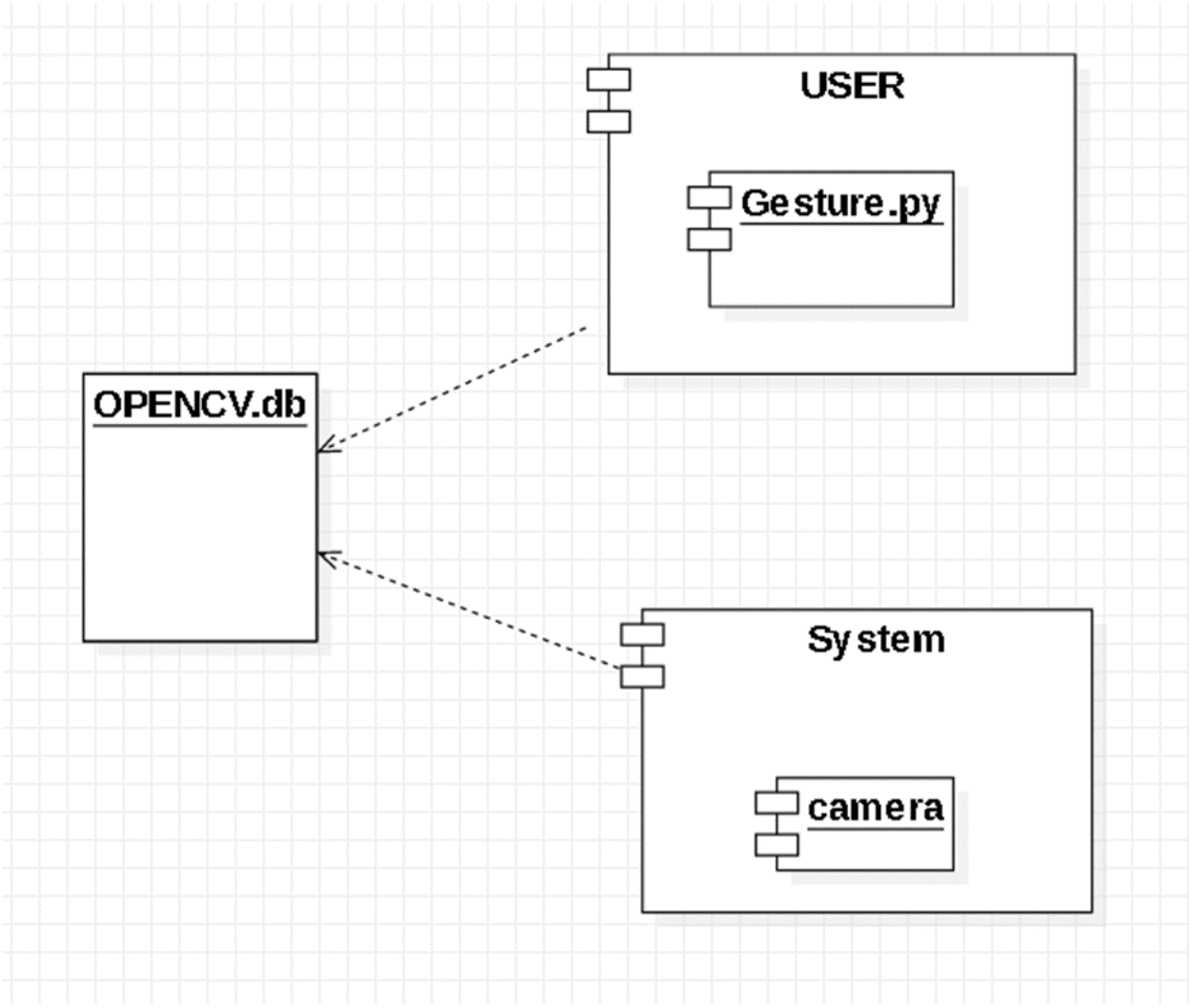

FIGURE 4.3.2.7: COMPONENT DIAGRAM

5. INPUT/OUTPUT DESIGN

5.1 INPUT DESIGN:

Input Design plays a vital role in the life cycle of software development, it requires very careful attention of developers. The input design is to feed data to the application as accurate as possible. So inputs are supposed to be designed effectively so that the errors occurring while feeding are minimized. According to Software Engineering Concepts, the input forms or screens are designed to provide to have a validation control over the input limit, range and other related validations.Input design is the process of converting the user created input into a computer-based format. The goal of the input design is to make the data entry logical and free from errors.

Validations are required for each data entered. Whenever a user enters an erroneous data, error message is displayed and the user can move on to the subsequent pages after completing all the entries in the current page.

5.2 OUTPUT DESIGN:

The Output from the computer is required to mainly create an efficient method of communication within the company primarily among the project leader and his team members, in other words, the administrator and the clients. The output of VPN is the system which allows the project leader to manage his clients in terms of creating new clients and assigning new projects to them, maintaining a record of the project validity and providing folder level access to each client on the user side depending on the projects allotted to him. After completion of a project, a new project may be assigned to the client. User authentication procedures are maintained at the initial stages itself.

6. IMPLEMENTATION

6.1 GestureController.py File:

```
# Imports

import cv2
import mediapipe as mp
import pyautogui
import math
from enum import IntEnum
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from google.protobuf.json_format import MessageToDict
import screen_brightness_control as sbcontrol

pyautogui.FAILSAFE = False
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands

# Gesture Encodings
class Gest(IntEnum):
    # Binary Encoded
    FIST = 0
    PINKY = 1
    RING = 2
    MID = 4
    LAST3 = 7
    INDEX = 8
    FIRST2 = 12
    LAST4 = 15
    THUMB = 16
    PALM = 31

    # Extra Mappings
    V_GEST = 33
    TWO_FINGER_CLOSED = 34
    PINCH_MAJOR = 35
    PINCH_MINOR = 36

# Multi-handedness Labels
class HLabel(IntEnum):
    MINOR = 0
    MAJOR = 1
```

```
# Convert Mediapipe Landmarks to recognizable Gestures
class HandRecog:

    def __init__(self, hand_label):
        self.finger = 0
        self.ori_gesture = Gest.PALM
        self.prev_gesture = Gest.PALM
        self.frame_count = 0
        self.hand_result = None
        self.hand_label = hand_label

    def update_hand_result(self, hand_result):
        self.hand_result = hand_result

    def get_signed_dist(self, point):
        sign = -1
        if self.hand_result.landmark[point[0]].y <
self.hand_result.landmark[point[1]].y:
            sign = 1
        dist = (self.hand_result.landmark[point[0]].x -
self.hand_result.landmark[point[1]].x)**2
        dist += (self.hand_result.landmark[point[0]].y -
self.hand_result.landmark[point[1]].y)**2
        dist = math.sqrt(dist)
        return dist*sign

    def get_dist(self, point):
        dist = (self.hand_result.landmark[point[0]].x -
self.hand_result.landmark[point[1]].x)**2
        dist += (self.hand_result.landmark[point[0]].y -
self.hand_result.landmark[point[1]].y)**2
        dist = math.sqrt(dist)
        return dist

    def get_dz(self,point):
        return abs(self.hand_result.landmark[point[0]].z -
self.hand_result.landmark[point[1]].z)

    # Function to find Gesture Encoding using current finger_state.
    # Finger_state: 1 if finger is open, else 0
    def set_finger_state(self):
        if self.hand_result == None:
            return

        points = [[8,5,0],[12,9,0],[16,13,0],[20,17,0]]
        self.finger = 0
```

```
            self.finger = self.finger | 0 #thumb
            for idx,point in enumerate(points):

                dist = self.get_signed_dist(point[:2])
                dist2 = self.get_signed_dist(point[1:])

                try:
                    ratio = round(dist/dist2,1)
                except:
                    ratio = round(dist1/0.01,1)

                self.finger = self.finger << 1
                if ratio > 0.5 :
                    self.finger = self.finger | 1

        # Handling Fluctations due to noise
        def get_gesture(self):
            if self.hand_result == None:
                return Gest.PALM

            current_gesture = Gest.PALM
            if self.finger in [Gest.LAST3,Gest.LAST4] and self.get_dist([8,4]) <
    0.05:
                if self.hand_label == HLabel.MINOR :
                    current_gesture = Gest.PINCH_MINOR
                else:
                    current_gesture = Gest.PINCH_MAJOR

            elif Gest.FIRST2 == self.finger :
                point = [[8,12],[5,9]]
                dist1 = self.get_dist(point[0])
                dist2 = self.get_dist(point[1])
                ratio = dist1/dist2
                if ratio > 1.7:
                    current_gesture = Gest.V_GEST
                else:
                    if self.get_dz([8,12]) < 0.1:
                        current_gesture =  Gest.TWO_FINGER_CLOSED
                    else:
                        current_gesture =  Gest.MID

            else:
                current_gesture =  self.finger

            if current_gesture == self.prev_gesture:
                self.frame_count += 1
```

```
        else:
            self.frame_count = 0

        self.prev_gesture = current_gesture

        if self.frame_count > 4 :
            self.ori_gesture = current_gesture
        return self.ori_gesture

# Executes commands according to detected gestures
class Controller:
    tx_old = 0
    ty_old = 0
    trial = True
    flag = False
    grabflag = False
    pinchmajorflag = False
    pinchminorflag = False
    pinchstartxcoord = None
    pinchstartycoord = None
    pinchdirectionflag = None
    prevpinchlv = 0
    pinchlv = 0
    framecount = 0
    prev_hand = None
    pinch_threshold = 0.3

    def getpinchylv(hand_result):
        dist = round((Controller.pinchstartycoord -
hand_result.landmark[8].y)*10,1)
        return dist

    def getpinchxlv(hand_result):
        dist = round((hand_result.landmark[8].x -
Controller.pinchstartxcoord)*10,1)
        return dist

    def changesystembrightness():
        currentBrightnessLv = sbcontrol.get_brightness()/100.0
        currentBrightnessLv += Controller.pinchlv/50.0
        if currentBrightnessLv > 1.0:
            currentBrightnessLv = 1.0
        elif currentBrightnessLv < 0.0:
            currentBrightnessLv = 0.0
        sbcontrol.fade_brightness(int(100*currentBrightnessLv) , start =
sbcontrol.get_brightness())
```

```
    def changesystemvolume():
        devices = AudioUtilities.GetSpeakers()
        interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL,
None)
        volume = cast(interface, POINTER(IAudioEndpointVolume))
        currentVolumeLv = volume.GetMasterVolumeLevelScalar()
        currentVolumeLv += Controller.pinchlv/50.0
        if currentVolumeLv > 1.0:
            currentVolumeLv = 1.0
        elif currentVolumeLv < 0.0:
            currentVolumeLv = 0.0
        volume.SetMasterVolumeLevelScalar(currentVolumeLv, None)

    def scrollVertical():
        pyautogui.scroll(120 if Controller.pinchlv>0.0 else -120)


    def scrollHorizontal():
        pyautogui.keyDown('shift')
        pyautogui.keyDown('ctrl')
        pyautogui.scroll(-120 if Controller.pinchlv>0.0 else 120)
        pyautogui.keyUp('ctrl')
        pyautogui.keyUp('shift')

    # Locate Hand to get Cursor Position
    # Stabilize cursor by Dampening
    def get_position(hand_result):
        point = 9
        position = [hand_result.landmark[point].x
,hand_result.landmark[point].y]
        sx,sy = pyautogui.size()
        x_old,y_old = pyautogui.position()
        x = int(position[0]*sx)
        y = int(position[1]*sy)
        if Controller.prev_hand is None:
            Controller.prev_hand = x,y
        delta_x = x - Controller.prev_hand[0]
        delta_y = y - Controller.prev_hand[1]

        distsq = delta_x**2 + delta_y**2
        ratio = 1
        Controller.prev_hand = [x,y]

        if distsq <= 25:
            ratio = 0
```

```
        elif distsq <= 900:
            ratio = 0.07 * (distsq ** (1/2))
        else:
            ratio = 2.1
        x , y = x_old + delta_x*ratio , y_old + delta_y*ratio
        return (x,y)

    def pinch_control_init(hand_result):
        Controller.pinchstartxcoord = hand_result.landmark[8].x
        Controller.pinchstartycoord = hand_result.landmark[8].y
        Controller.pinchlv = 0
        Controller.prevpinchlv = 0
        Controller.framecount = 0

    # Hold final position for 5 frames to change status
    def pinch_control(hand_result, controlHorizontal, controlVertical):
        if Controller.framecount == 5:
            Controller.framecount = 0
            Controller.pinchlv = Controller.prevpinchlv

            if Controller.pinchdirectionflag == True:
                controlHorizontal() #x

            elif Controller.pinchdirectionflag == False:
                controlVertical() #y

        lvx =  Controller.getpinchxlv(hand_result)
        lvy =  Controller.getpinchylv(hand_result)

        if abs(lvy) > abs(lvx) and abs(lvy) > Controller.pinch_threshold:
            Controller.pinchdirectionflag = False
            if abs(Controller.prevpinchlv - lvy) < Controller.pinch_threshold:
                Controller.framecount += 1
            else:
                Controller.prevpinchlv = lvy
                Controller.framecount = 0

        elif abs(lvx) > Controller.pinch_threshold:
            Controller.pinchdirectionflag = True
            if abs(Controller.prevpinchlv - lvx) < Controller.pinch_threshold:
                Controller.framecount += 1
            else:
                Controller.prevpinchlv = lvx
                Controller.framecount = 0

    def handle_controls(gesture, hand_result):
```

```
            x,y = None,None
            if gesture != Gest.PALM :
                x,y = Controller.get_position(hand_result)

            # flag reset
            if gesture != Gest.FIST and Controller.grabflag:
                Controller.grabflag = False
                pyautogui.mouseUp(button = "left")

            if gesture != Gest.PINCH_MAJOR and Controller.pinchmajorflag:
                Controller.pinchmajorflag = False

            if gesture != Gest.PINCH_MINOR and Controller.pinchminorflag:
                Controller.pinchminorflag = False

            # implementation
            if gesture == Gest.V_GEST:
                Controller.flag = True
                pyautogui.moveTo(x, y, duration = 0.1)

            elif gesture == Gest.FIST:
                if not Controller.grabflag :
                    Controller.grabflag = True
                    pyautogui.mouseDown(button = "left")
                pyautogui.moveTo(x, y, duration = 0.1)

            elif gesture == Gest.MID and Controller.flag:
                pyautogui.click()
                Controller.flag = False

            elif gesture == Gest.INDEX and Controller.flag:
                pyautogui.click(button='right')
                Controller.flag = False

            elif gesture == Gest.TWO_FINGER_CLOSED and Controller.flag:
                pyautogui.doubleClick()
                Controller.flag = False

            elif gesture == Gest.PINCH_MINOR:
                if Controller.pinchminorflag == False:
                    Controller.pinch_control_init(hand_result)
                    Controller.pinchminorflag = True
                Controller.pinch_control(hand_result,Controller.scrollHorizontal,
        Controller.scrollVertical)

            elif gesture == Gest.PINCH_MAJOR:
```

```
            if Controller.pinchmajorflag == False:
                Controller.pinch_control_init(hand_result)
                Controller.pinchmajorflag = True

Controller.pinch_control(hand_result,Controller.changesystembrightness,
Controller.changesystemvolume)

'''
----------------------------------------Main Class----------------------------
------------
    Entry point of Gesture Controller
'''


class GestureController:
    gc_mode = 0
    cap = None
    CAM_HEIGHT = None
    CAM_WIDTH = None
    hr_major = None # Right Hand by default
    hr_minor = None # Left hand by default
    dom_hand = True

    def __init__(self):
        GestureController.gc_mode = 1
        GestureController.cap = cv2.VideoCapture(0)
        GestureController.CAM_HEIGHT =
GestureController.cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
        GestureController.CAM_WIDTH =
GestureController.cap.get(cv2.CAP_PROP_FRAME_WIDTH)

    def classify_hands(results):
        left , right = None,None
        try:
            handedness_dict = MessageToDict(results.multi_handedness[0])
            if handedness_dict['classification'][0]['label'] == 'Right':
                right = results.multi_hand_landmarks[0]
            else :
                left = results.multi_hand_landmarks[0]
        except:
            pass

        try:
            handedness_dict = MessageToDict(results.multi_handedness[1])
            if handedness_dict['classification'][0]['label'] == 'Right':
                right = results.multi_hand_landmarks[1]
```

```
            else :
                left = results.multi_hand_landmarks[1]
        except:
            pass

        if GestureController.dom_hand == True:
            GestureController.hr_major = right
            GestureController.hr_minor = left
        else :
            GestureController.hr_major = left
            GestureController.hr_minor = right

    def start(self):

        handmajor = HandRecog(HLabel.MAJOR)
        handminor = HandRecog(HLabel.MINOR)

        with mp_hands.Hands(max_num_hands = 2,min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
            while GestureController.cap.isOpened() and
GestureController.gc_mode:
                success, image = GestureController.cap.read()

                if not success:
                    print("Ignoring empty camera frame.")
                    continue

                image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
                image.flags.writeable = False
                results = hands.process(image)

                image.flags.writeable = True
                image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

                if results.multi_hand_landmarks:
                    GestureController.classify_hands(results)
                    handmajor.update_hand_result(GestureController.hr_major)
                    handminor.update_hand_result(GestureController.hr_minor)

                    handmajor.set_finger_state()
                    handminor.set_finger_state()
                    gest_name = handminor.get_gesture()

                    if gest_name == Gest.PINCH_MINOR:
                        Controller.handle_controls(gest_name,
handminor.hand_result)
```

```
                    else:
                        gest_name = handmajor.get_gesture()
                        Controller.handle_controls(gest_name,
handmajor.hand_result)

                    for hand_landmarks in results.multi_hand_landmarks:
                        mp_drawing.draw_landmarks(image, hand_landmarks,
mp_hands.HAND_CONNECTIONS)
                else:
                    Controller.prev_hand = None
                cv2.imshow('Gesture Controller', image)
                if cv2.waitKey(5) & 0xFF == 13:
                    break
        GestureController.cap.release()
        cv2.destroyAllWindows()

# uncomment to run directly
gc1 = GestureController()
gc1.start()
```

6.2 Gesture_handRecog.py File:

```
import numpy as np
import cv2
import cv2.aruco as aruco
import os
import glob
import math
import pyautogui
import time

class Marker:
    def __init__(self, dict_type = aruco.DICT_4X4_50, thresh_constant = 1):
        self.aruco_dict = aruco.Dictionary_get(dict_type)
        self.parameters = aruco.DetectorParameters_create()
        self.parameters.adaptiveThreshConstant = thresh_constant
        self.corners = None # corners of Marker
        self.marker_x2y = 1 # width:height ratio
        self.mtx, self.dist = Marker.calibrate()

    def calibrate():
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30,
0.001)
        objp = np.zeros((6*7,3), np.float32)
        objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)
        objpoints = [] # 3d point in real world space
```

```
        imgpoints = [] # 2d points in image plane.
        path = os.path.dirname(os.path.abspath(__file__))
        p1 = path + r'\calib_images\checkerboard\*.jpg'
        images = glob.glob(p1)
        for fname in images:
            img = cv2.imread(fname)
            gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            ret, corners = cv2.findChessboardCorners(gray, (7,6),None)
            if ret == True:
                objpoints.append(objp)
                corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-
1),criteria)
                imgpoints.append(corners2)
                img = cv2.drawChessboardCorners(img, (7,6), corners2,ret)

        ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,
imgpoints, gray.shape[::-1],None,None)

        #mtx =
[[534.34144579,0.0,339.15527836],[0.0,534.68425882,233.84359493],[0.0,0.0,1.0]
]
        #dist = [[-2.88320983e-01, 5.41079685e-02, 1.73501622e-03, -
2.61333895e-04, 2.04110465e-01]]
        return mtx, dist

    def detect(self, frame):
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        self.corners, ids, rejectedImgPoints = aruco.detectMarkers(gray_frame,
self.aruco_dict, parameters = self.parameters)
        if np.all(ids != None):
            rvec, tvec ,_ = aruco.estimatePoseSingleMarkers(self.corners,
0.05, self.mtx, self.dist)
        else:
            self.corners = None

    def is_detected(self):
        if self.corners:
            return True
        return False

    def draw_marker(self, frame):
        aruco.drawDetectedMarkers(frame, self.corners)

def ecu_dis(p1, p2):
    dist = np.sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)
    return dist
```

```
def find_HSV(samples):
    try:
        color = np.uint8([ samples ])
    except:
        color = np.uint8([ [[105,105,50]] ])
    hsv_color = cv2.cvtColor(color,cv2.COLOR_RGB2HSV)
    #print( hsv_color )
    return hsv_color

def draw_box(frame, points, color=(0,255,127)):
    if points:
        frame = cv2.line(frame, points[0], points[1], color, thickness=2,
lineType=8) #top
        frame = cv2.line(frame, points[1], points[2], color, thickness=2,
lineType=8) #right
        frame = cv2.line(frame, points[2], points[3], color, thickness=2,
lineType=8) #bottom
        frame = cv2.line(frame, points[3], points[0], color, thickness=2,
lineType=8) #left

def in_cam(val, type_):
    if type_ == 'x':
        if val<0:
            return 0
        if val>GestureController.cam_width:
            return GestureController.cam_width
    elif type_ == 'y':
        if val<0:
            return 0
        if val>GestureController.cam_height:
            return GestureController.cam_height
    return val


class ROI:
    def __init__(self, roi_alpha1=1.5, roi_alpha2=1.5, roi_beta=2.5, hsv_alpha
= 0.3, hsv_beta = 0.5, hsv_lift_up = 0.3):
        self.roi_alpha1 = roi_alpha1
        self.roi_alpha2 = roi_alpha2
        self.roi_beta = roi_beta
        self.roi_corners = None

        self.hsv_alpha = hsv_alpha
        self.hsv_beta = hsv_beta
        self.hsv_lift_up = hsv_lift_up
        self.hsv_corners = None
```

```
        self.marker_top = None
        self.glove_hsv = None

    def findROI(self, frame, marker):
        rec_coor = marker.corners[0][0]
        c1 = (int(rec_coor[0][0]),int(rec_coor[0][1]))
        c2 = (int(rec_coor[1][0]),int(rec_coor[1][1]))
        c3 = (int(rec_coor[2][0]),int(rec_coor[2][1]))
        c4 = (int(rec_coor[3][0]),int(rec_coor[3][1]))

        try:
            marker.marker_x2y = np.sqrt((c1[0]-c2[0])**2 + (c1[1]-c2[1])**2) /
np.sqrt((c3[0]-c2[0])**2 + (c3[1]-c2[1])**2)
        except:
            marker.marker_x2y = 999.0

        #mid-point of top line of Marker
        cx = (c1[0] + c2[0])/2
        cy = (c1[1] + c2[1])/2

        self.marker_top = [cx, cy]

        l = np.absolute(ecu_dis(c1,c4))

        try:
            slope_12 = (c1[1]-c2[1])/(c1[0]-c2[0])
        except:
            slope_12 = (c1[1]-c2[1])*999.0 + 0.1

        try:
            slope_14 = -1 / slope_12
        except:
            slope_14 = -999.0

        if slope_14 < 0:
            sign = 1
        else:
            sign = -1

        bot_rx = int(cx + self.roi_alpha2 * l * np.sqrt(1/(1+slope_12**2)))
        bot_ry = int(cy + self.roi_alpha2 * slope_12 * l *
np.sqrt(1/(1+slope_12**2)))

        bot_lx = int(cx - self.roi_alpha1 * l * np.sqrt(1/(1+slope_12**2)))
        bot_ly = int(cy - self.roi_alpha1 * slope_12 * l *
np.sqrt(1/(1+slope_12**2)))
```

```
        top_lx = int(bot_lx + sign * self.roi_beta * l *
np.sqrt(1/(1+slope_14**2)))
        top_ly = int(bot_ly + sign * self.roi_beta * slope_14 * l *
np.sqrt(1/(1+slope_14**2)))

        top_rx = int(bot_rx + sign * self.roi_beta * l *
np.sqrt(1/(1+slope_14**2)))
        top_ry = int(bot_ry + sign * self.roi_beta * slope_14 * l *
np.sqrt(1/(1+slope_14**2)))

        bot_lx = in_cam(bot_lx, 'x')
        bot_ly = in_cam(bot_ly, 'y')

        bot_rx = in_cam(bot_rx, 'x')
        bot_ry = in_cam(bot_ry, 'y')

        top_lx = in_cam(top_lx, 'x')
        top_ly = in_cam(top_ly, 'y')

        top_rx = in_cam(top_rx, 'x')
        top_ry = in_cam(top_ry, 'y')

        self.roi_corners = [(bot_lx,bot_ly), (bot_rx,bot_ry), (top_rx,top_ry),
(top_lx,top_ly)]


    def find_glove_hsv(self, frame, marker):
        rec_coor = marker.corners[0][0]
        c1 = (int(rec_coor[0][0]),int(rec_coor[0][1]))
        c2 = (int(rec_coor[1][0]),int(rec_coor[1][1]))
        c3 = (int(rec_coor[2][0]),int(rec_coor[2][1]))
        c4 = (int(rec_coor[3][0]),int(rec_coor[3][1]))

        l = np.absolute(ecu_dis(c1,c4))

        try:
            slope_12 = (c1[1]-c2[1])/(c1[0]-c2[0])
        except:
            slope_12 = (c1[1]-c2[1])*999.0 + 0.1
        try:
            slope_14 = -1 / slope_12
        except:
            slope_14 = -999.0

        if slope_14 < 0:
            sign = 1
```

```
        else:
            sign = -1

        bot_rx = int(self.marker_top[0] + self.hsv_alpha * l *
np.sqrt(1/(1+slope_12**2)))
        bot_ry = int(self.marker_top[1] - self.hsv_lift_up*l + self.hsv_alpha
* slope_12 * l * np.sqrt(1/(1+slope_12**2)))

        bot_lx = int(self.marker_top[0] - self.hsv_alpha * l *
np.sqrt(1/(1+slope_12**2)))
        bot_ly = int(self.marker_top[1] - self.hsv_lift_up*l - self.hsv_alpha
* slope_12 * l * np.sqrt(1/(1+slope_12**2)))

        top_lx = int(bot_lx + sign * self.hsv_beta * l *
np.sqrt(1/(1+slope_14**2)))
        top_ly = int(bot_ly + sign * self.hsv_beta * slope_14 * l *
np.sqrt(1/(1+slope_14**2)))

        top_rx = int(bot_rx + sign * self.hsv_beta * l *
np.sqrt(1/(1+slope_14**2)))
        top_ry = int(bot_ry + sign * self.hsv_beta * slope_14 * l *
np.sqrt(1/(1+slope_14**2)))

        region = frame[top_ry:bot_ry , top_lx:bot_rx]
        b, g, r = np.mean(region, axis=(0, 1))

        self.hsv_glove = find_HSV([[r,g,b]])
        self.hsv_corners =  [(bot_lx,bot_ly), (bot_rx,bot_ry),
(top_rx,top_ry), (top_lx,top_ly)]


    def cropROI(self, frame):
        pts = np.array(self.roi_corners)

        ## (1) Crop the bounding rect
        rect = cv2.boundingRect(pts)
        x,y,w,h = rect
        croped = frame[y:y+h, x:x+w].copy()

        ## (2) make mask
        pts = pts - pts.min(axis=0)

        mask = np.zeros(croped.shape[:2], np.uint8)
        cv2.drawContours(mask, [pts], -1, (255, 255, 255), -1, cv2.LINE_AA)

        ## (3) do bit-op
```

```
        dst = cv2.bitwise_and(croped, croped, mask=mask)

        ## (4) add the white background
        bg = np.ones_like(croped, np.uint8)*255
        cv2.bitwise_not(bg,bg, mask=mask)

        kernelOpen = np.ones((3,3),np.uint8)
        kernelClose = np.ones((5,5),np.uint8)

        hsv = cv2.cvtColor(dst, cv2.COLOR_BGR2HSV)

        lower_range = np.array([self.hsv_glove[0][0][0]//1-5,50,50])
        upper_range = np.array([self.hsv_glove[0][0][0]//1+5,255,255])

        mask = cv2.inRange(hsv, lower_range, upper_range)
        #mask = cv2.dilate(mask,kernelOpen,iterations = 1)
        Opening =cv2.morphologyEx(mask,cv2.MORPH_OPEN,kernelOpen)
        Closing =cv2.morphologyEx(Opening,cv2.MORPH_CLOSE,kernelClose)
        FinalMask = Closing

        return FinalMask
class Glove:

    def __init__(self):
        self.fingers = 0
        self.arearatio = 0
        self.gesture = 0

    def find_fingers(self, FinalMask):

conts,h=cv2.findContours(FinalMask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
        hull = [cv2.convexHull(c) for c in conts]

        try:
            cnt = max(conts, key = lambda x: cv2.contourArea(x))
            #approx the contour a little
            epsilon = 0.0005*cv2.arcLength(cnt,True)
            approx= cv2.approxPolyDP(cnt,epsilon,True)
            #make convex hull around hand
            hull = cv2.convexHull(cnt)
            #define area of hull and area of hand
            areahull = cv2.contourArea(hull)
            areacnt = cv2.contourArea(cnt)
            #find the percentage of area not covered by hand in convex hull
            self.arearatio=((areahull-areacnt)/areacnt)*100
            #find the defects in convex hull with respect to hand
```

```
            hull = cv2.convexHull(approx, returnPoints=False)
            defects = cv2.convexityDefects(approx, hull)
        except:
            print("No Contours found in FinalMask")

        # l = no. of defects
        l=0
        try:
            #code for finding no. of defects due to fingers
            for i in range(defects.shape[0]):
                s,e,f,d = defects[i,0]
                start = tuple(approx[s][0])
                end = tuple(approx[e][0])
                far = tuple(approx[f][0])

                # find length of all sides of triangle
                a = math.sqrt((end[0] - start[0])**2 + (end[1] - start[1])**2)
                b = math.sqrt((far[0] - start[0])**2 + (far[1] - start[1])**2)
                c = math.sqrt((end[0] - far[0])**2 + (end[1] - far[1])**2)
                s = (a+b+c)/2
                ar = math.sqrt(s*(s-a)*(s-b)*(s-c))

                #distance between point and convex hull
                d=(2*ar)/a

                # apply cosine rule here
                angle = math.acos((b**2 + c**2 - a**2)/(2*b*c)) * 57

                # ignore angles > 90 and ignore points very close to convex
hull(they generally come due to noise)
                if angle <= 90 and d>30:
                    l += 1
                    #cv2.circle(frame, far, 3, [255,255,255], -1)

                #draw lines around hand
                cv2.line(FinalMask,start, end, [255,255,255], 2)

            l+=1
        except:
            l = 0
            print("No Defects found in mask")
        self.fingers = l

    def find_gesture(self, frame):
        font = cv2.FONT_HERSHEY_SIMPLEX
        self.gesture = 0
```

```
        if self.fingers==1:
            #cv2.putText(frame, str(int(arearatio)), (10,50), font, 2,
(0,0,255), 3, cv2.LINE_AA)
            if self.arearatio<15:
                cv2.putText(frame,'0',(0,50), font, 2, (0,0,255), 3,
cv2.LINE_AA)
                self.gesture = 0
            elif self.arearatio<25:
                cv2.putText(frame,'2 fingers',(0,50), font, 2, (0,0,255), 3,
cv2.LINE_AA)
                self.gesture = 2
            else:
                cv2.putText(frame,'1 finger',(0,50), font, 2, (0,0,255), 3,
cv2.LINE_AA)
                self.gesture = 1

        elif self.fingers==2:
            cv2.putText(frame,'2', (0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)
            self.gesture = 3
        '''
        elif self.fingers==3:
            #cv2.putText(frame,'3',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

        elif self.fingers==4:
            #cv2.putText(frame,'4',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

        elif self.fingers==5:
            #cv2.putText(frame,'5',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

        else:
           # cv2.putText(frame,'reposition',(10,50), font, 2, (0,0,255), 3,
cv2.LINE_AA)
class Tracker:
    def __init__(self):
        self.tracker_started = False
        self.tracker = None
        self.start_time = 0.0
        self.now_time = 0.0
        self.tracker_bbox = None

    def corners_to_tracker(self, corners):
        csrt_minX = int( min( [corners[0][0][0][0], corners[0][0][1][0],
corners[0][0][2][0], corners[0][0][3][0]] ))
        csrt_maxX = int( max( [corners[0][0][0][0], corners[0][0][1][0],
corners[0][0][2][0], corners[0][0][3][0]] ))
```

```
        csrt_minY = int( min( [corners[0][0][0][1], corners[0][0][1][1],
corners[0][0][2][1], corners[0][0][3][1]] ))
        csrt_maxY = int( max( [corners[0][0][0][1], corners[0][0][1][1],
corners[0][0][2][1], corners[0][0][3][1]] ))
        self.tracker_bbox = [csrt_minX, csrt_minY, csrt_maxX-csrt_minX,
csrt_maxY-csrt_minY]
    def tracker_to_corner(self, final_bbox):
        if self.tracker_bbox == None:
            return None
        final_bbox = [[[1,2],[3,4],[5,6],[7,8]]]
        final_bbox[0][0] = [self.tracker_bbox[0],self.tracker_bbox[1]]
        final_bbox[0][1] = [self.tracker_bbox[0]+
self.tracker_bbox[2],self.tracker_bbox[1]]
        final_bbox[0][2] = [self.tracker_bbox[0]+
self.tracker_bbox[2],self.tracker_bbox[1] + self.tracker_bbox[3]]
        final_bbox[0][3] = [self.tracker_bbox[0],self.tracker_bbox[1]
+self.tracker_bbox[3]]
        return [np.array(final_bbox, dtype = 'f')]

    def CSRT_tracker(self, frame):
        if self.tracker_bbox == None and self.tracker_started == False:
            return

        if self.tracker_started == False:
            if self.tracker == None:
                self.tracker = cv2.TrackerCSRT_create()

        if self.tracker_bbox != None:
            try:
                self.start_time = time.time()
                ok = self.tracker.init(frame, self.tracker_bbox)
                self.tracker_started = True
            except:
                print("tracker.init failed")
        try:
            ok, self.tracker_bbox = self.tracker.update(frame)
        except:
            ok = None
            print("tracker.update failed")
        self.now_time = time.time()

        if self.now_time-self.start_time >= 2.0 :
#cv2.putText(frame, "Please posture your hand correctly", (10,50),
cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,255),1)
            cv2.putText(frame,'Posture your hand correctly',(10,10),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,0,255), 1, cv2.LINE_AA)
```

```
            #print("tracking timeout")
            self.tracker_started = False
            self.tracker_bbox = None
            return
        if ok:
            # Tracking success
            p1 = (int(self.tracker_bbox[0]), int(self.tracker_bbox[1]))
            p2 = (int(self.tracker_bbox[0] + self.tracker_bbox[2]),
int(self.tracker_bbox[1] + self.tracker_bbox[3]))
            cv2.rectangle(frame, p1, p2, (80, 255, 255), 2, 1)
        else :
            # Tracking failure
            self.tracker_started = False
            cv2.putText(frame, "Tracking failure detected", (100,80),
cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2)
            print("Tracking failure detected")
            #reintiallize code to tackle tracking failure
class Mouse:
    def __init__(self):
        self.tx_old = 0
        self.ty_old = 0
        self.trial = True
        self.flag = 0

    def move_mouse(self,frame,position,gesture):

        (sx,sy)=pyautogui.size()
        (camx,camy) = (frame.shape[:2][0],frame.shape[:2][1])
        (mx_old,my_old) = pyautogui.position()
        Damping = 2 # Hyperparameter we will have to adjust
        tx = position[0]
        ty = position[1]
        if self.trial:
            self.trial, self.tx_old, self.ty_old = False, tx, ty
        delta_tx = tx - self.tx_old
        delta_ty = ty - self.ty_old
        self.tx_old,self.ty_old = tx,ty

        if (gesture == 3):
            self.flag = 0
            mx = mx_old + (delta_tx*sx) // (camx*Damping)
            my = my_old + (delta_ty*sy) // (camy*Damping)
            pyautogui.moveTo(mx,my, duration = 0.1)

        elif(gesture == 0):
            if self.flag == 0:
```

```
                pyautogui.doubleClick()
                self.flag = 1
        elif(gesture == 1):
            print('1 Finger Open')
class GestureController:
    gc_mode = 0
    pyautogui.FAILSAFE = False
    f_start_time = 0
    f_now_time = 0
    cam_width  = 0
    cam_height = 0

    aru_marker = Marker()
    hand_roi = ROI(2.5, 2.5, 6, 0.45, 0.6, 0.4)
    glove = Glove()
    csrt_track = Tracker()
    mouse = Mouse()

    def __init__(self):
        GestureController.cap = cv2.VideoCapture(0)
        if GestureController.cap.isOpened():
            GestureController.cam_width  = int(
GestureController.cap.get(cv2.CAP_PROP_FRAME_WIDTH) )
            GestureController.cam_height = int(
GestureController.cap.get(cv2.CAP_PROP_FRAME_HEIGHT) )
        else:
            print("CANNOT OPEN CAMERA")
        GestureController.gc_mode = 1
        GestureController.f_start_time = time.time()
        GestureController.f_now_time = time.time()

    def start(self):
        while (True):
            #mode checking
            if not GestureController.gc_mode:
                print('Exiting Gesture Controller')
                break
            #fps control
            fps = 30.0
            GestureController.f_start_time = time.time()
            while (GestureController.f_now_time-GestureController.f_start_time
<= 1.0/fps):
                GestureController.f_now_time = time.time()
            #read camera
            ret, frame = GestureController.cap.read()
            frame = cv2.flip(frame, 1)
```

```
            #detect Marker, find ROI, find glove HSV, get FinalMask on glove
            GestureController.aru_marker.detect(frame)
            if GestureController.aru_marker.is_detected():

GestureController.csrt_track.corners_to_tracker(GestureController.aru_marker.c
orners)
                GestureController.csrt_track.CSRT_tracker(frame)

            else:
                GestureController.csrt_track.tracker_bbox = None
                GestureController.csrt_track.CSRT_tracker(frame)
                GestureController.aru_marker.corners =
GestureController.csrt_track.tracker_to_corner(GestureController.aru_marker.co
rners)

            if GestureController.aru_marker.is_detected():
                GestureController.hand_roi.findROI(frame,
GestureController.aru_marker)
                GestureController.hand_roi.find_glove_hsv(frame,
GestureController.aru_marker)
                FinalMask = GestureController.hand_roi.cropROI(frame)
                GestureController.glove.find_fingers(FinalMask)
                GestureController.glove.find_gesture(frame)

GestureController.mouse.move_mouse(frame,GestureController.hand_roi.marker_top
,GestureController.glove.gesture)
            #draw call
            if GestureController.aru_marker.is_detected():
                GestureController.aru_marker.draw_marker(frame)
                draw_box(frame, GestureController.hand_roi.roi_corners,
(255,0,0))
                draw_box(frame, GestureController.hand_roi.hsv_corners,
(0,0,250))
                cv2.imshow('FinalMask',FinalMask)

            #display frame
            cv2.imshow('frame',frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        # When everything done, release the capture
        GestureController.cap.release()
        cv2.destroyAllWindows()
```

7. TESTING

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation.

7.1 TESTING OBJECTIVES:

- To ensure that during operation the system will perform as per specification.
- To make sure that system meets the user requirements during operation
- To make sure that during the operation, incorrect input, processing and output will be detected
- To see that when correct inputs are fed to the system the outputs are correct
- To verify that the controls incorporated in the same system as intended
- Testing is a process of executing a program with the intent of finding an error
- A good test case is one that has a high probability of finding an as yet undiscovered error

The software developed has been tested successfully using the following testing strategies and any errors that are encountered are corrected and again the part of the program or the procedure or function is put to testing until all the errors are removed. A successful test is one that uncovers an as yet undiscovered error.

Note that the result of the system testing will prove that the system is working correctly. It will give confidence to system designer, users of the system, prevent frustration during implementation process etc.

7.2 TESTING METHODOLOGIES:

- ✓ White box testing.
- ✓ Black box testing.
- ✓ Unit testing.
- ✓ Integration testing.
- ✓ User acceptance testing.
- ✓ Output testing.
- ✓ Validation testing.
- ✓ System testing.

1) White Box Testing:

White box testing is a testing case design method that uses the control structure of the procedure design to derive test cases. All independents path in a module are exercised at least once, all logical decisions are exercised at once, execute all loops at boundaries and within their operational bounds exercise internal data structure to ensure their validity. Here the customer is given three chances to enter a valid choice out of the given menu. After which the control exits the current menu.

2) Black Box Testing:

Black Box Testing attempts to find errors in following areas or categories, incorrect or missing functions, interface error, errors in data structures, performance error and initialization and termination error. Here all the input data must match the data type to become a valid entry.

3) Unit Testing:

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a module's control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing.

4) Integration Testing:

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

The following are the types of Integration Testing:

- ✓ **Top Down Integration:**

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module.

- ✓ **Bottom Up Integration:**

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated.

5) User acceptance Testing:

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

6) Output Testing:

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated or displayed by the system under consideration. Hence the output format is considered in 2 ways – one is on screen and another in printed format.

7) Validation Testing:

Validation testing is generally performed on the following fields:

- ✓ **Text Field:**

The text field can contain only the number of characters lesser than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes and error message.

- ✓ **Numeric Field:**

The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error messages. The individual modules are checked for accuracy and what it has to perform.

- ✓ **Preparation of Test Data:**

Taking various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

- ✓ **Using Live Test Data:**

Live test data are those that are actually extracted from organization files. After a system is partially constructed, programmers or analysts often ask users to key in a set of data from their normal activities. Then, the systems person uses this data as a way to partially test the system. In other instances, programmers or analysts extract a set of live data from the files and have them entered themselves.

- ✓ **Using Artificial Test Data:**

Artificial test data are created solely for test purposes, since they can be generated to test all combinations of formats and values. In other words, the artificial data, which can quickly be prepared by a data generating utility program in the information systems department, make possible the testing of all login and control paths through the program.

The most effective test programs use artificial test data generated by persons other than those who wrote the programs. Often, an independent team of testers formulates a testing plan, using the systems specifications.

7.3 USER TRAINING:

Whenever a new system is developed, user training is required to educate them about the working of the system so that it can be put to efficient use by those for whom the system has been primarily designed. For this purpose the normal working of the project was demonstrated to the prospective users. Its working is easily understandable and since the expected users are people who have good knowledge of computers, the use of this system is very easy.

7.4 MAINTAINENCE:

This covers a wide range of activities including correcting code and design errors. To reduce the need for maintenance in the long run, we have more accurately defined the user's requirements during the process of system development. Depending on the requirements, this system has been developed to satisfy the needs to the largest possible extent. With development in technology, it may be possible to add many more features based on the requirements in future. The coding and designing is simple and easy to understand which will make maintenance easier.

7.5 TESTING STRATEGY :

A strategy for system testing integrates system test cases and design techniques into a well planned series of steps that results in the successful construction of software. The testing strategy must co-operate test planning, test case design, test execution, and the resultant data collection and evaluation .A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level tests that validate major system functions against user requirements.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding.

7.5.1 SYSTEM TESTING:

Software once validated must be combined with other system elements (e.g. Hardware, people, database). System testing verifies that all the elements are proper and that overall system function performance is achieved. It also tests to find discrepancies between the system and its original objective, current specifications and system documentation.

7.5.2 UNIT TESTING:

In unit testing different are modules are tested against the specifications produced during the design for the modules. Unit testing is essential for verification of the code produced during the coding phase, and hence the goals to test the internal logic of the modules. Using the detailed design description as a guide, important Conrail paths are tested to uncover errors within the boundary of the modules. This testing is carried out during the programming stage itself. In this type of testing step, each module was found to be working satisfactorily as regards to the expected output from the module. In Due Course, latest technology advancements will be taken into consideration. As part of technical build-up many components of the networking system will be generic in nature so that future projects can either use or interact with this.

8. OUTPUT SCREENS

8.1 NEUTRAL GESTURE:

Neutral Gesture is used to halt/stop execution of current gesture.

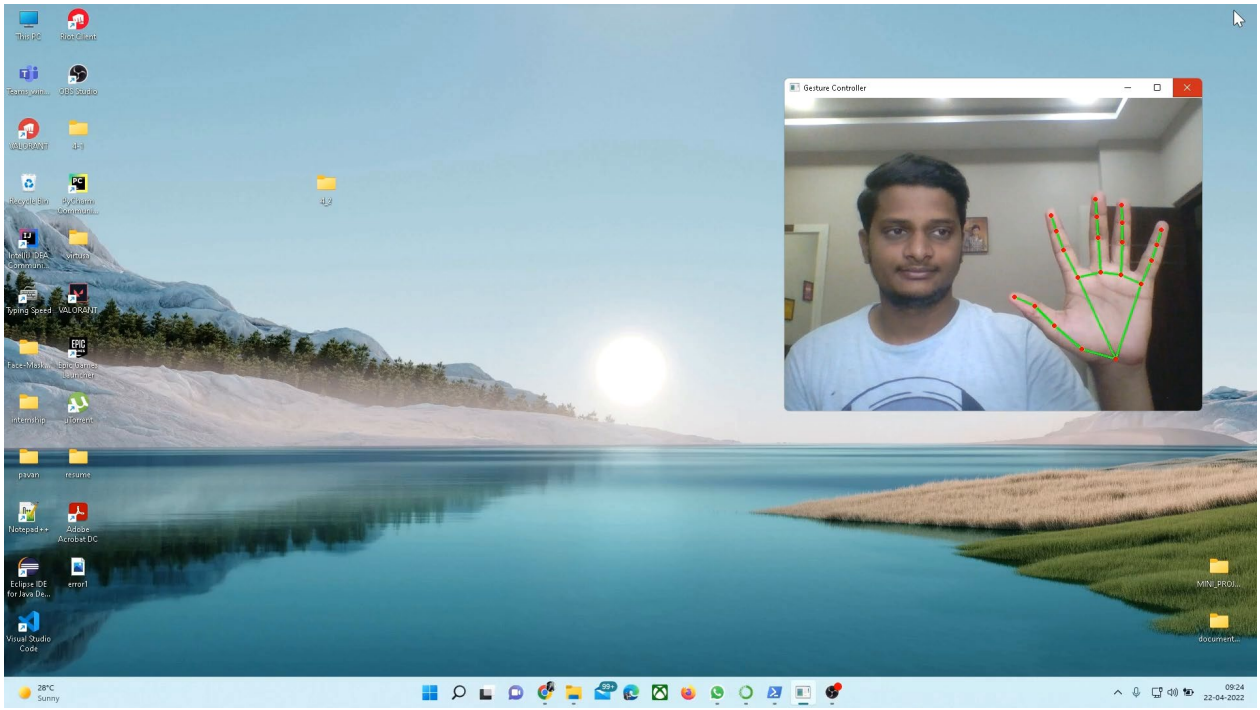

FIGURE 8.1: NEUTRAL GESTURE

8.2 MOVE CURSOR GESTURE:

Cursor is assigned to the midpoint of index and middle fingertips. This gesture moves the cursor to the desired location. Speed of the cursor movement is proportional to the speed of hand.

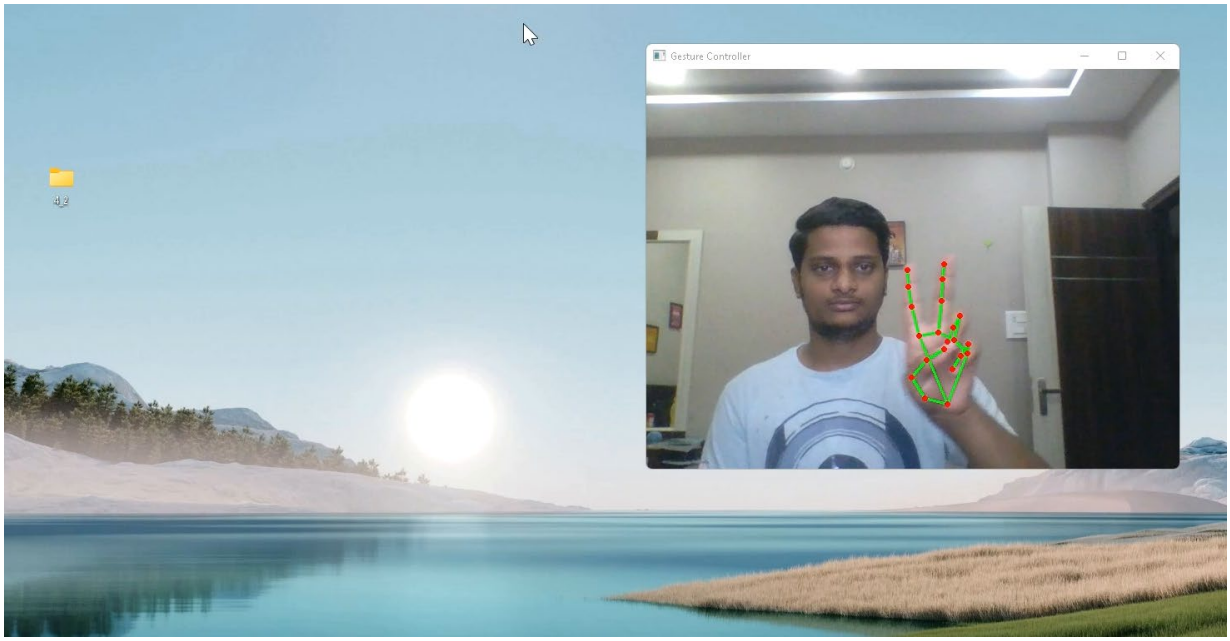

FIGURE 8.2: MOVE CURSOR

8.3 LEFT CLICK GESTURE:

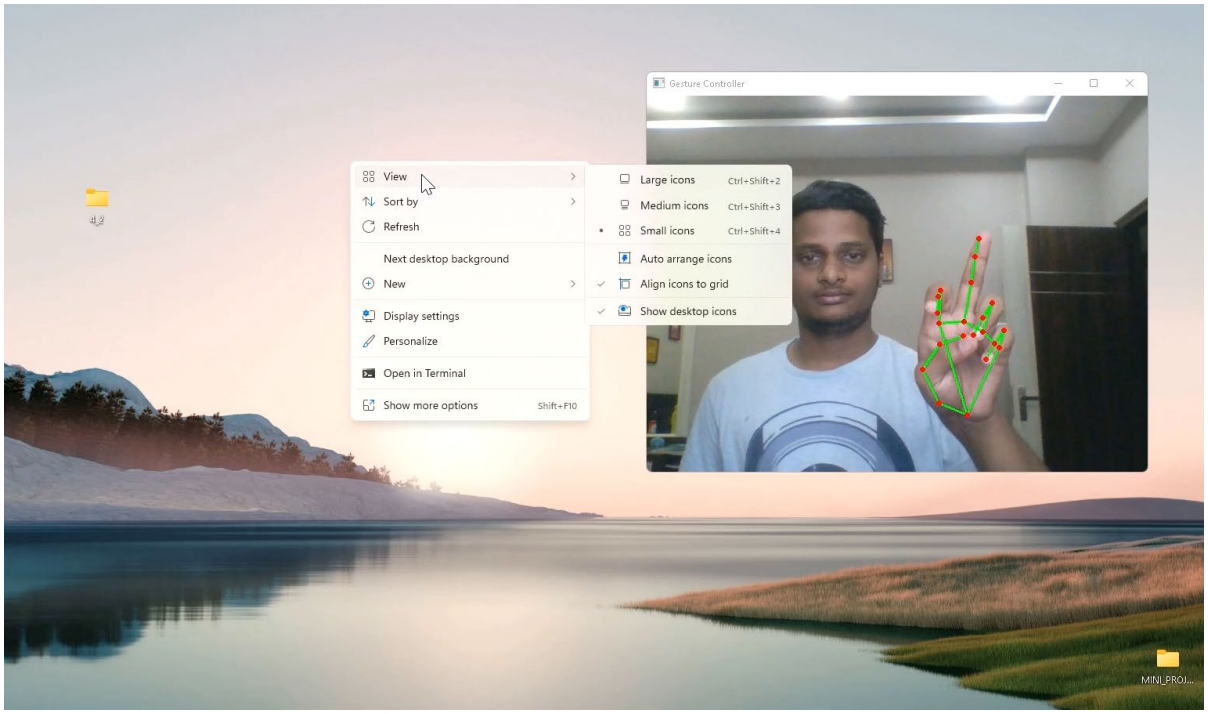

FIGURE 8.3: LEFT CLICK

8.4 RIGHT CLICK GESTURE:

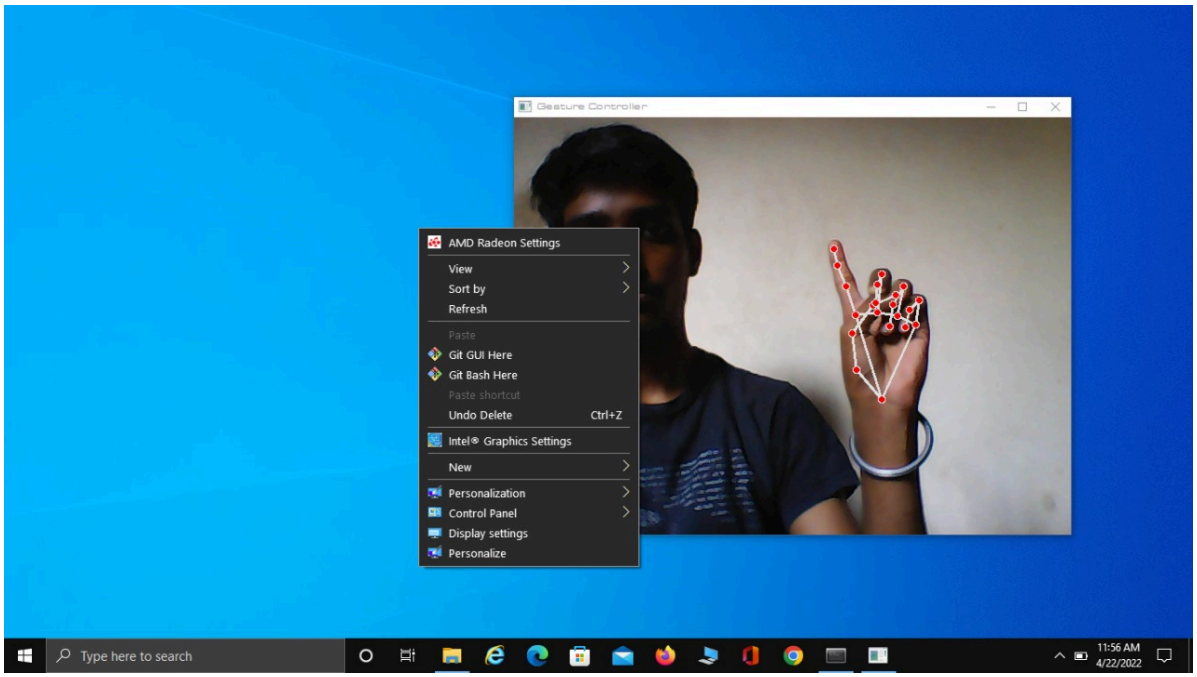

FIGURE 8.4: RIGHT CLICK

8.5 DOUBLE CLICK GESTURE:

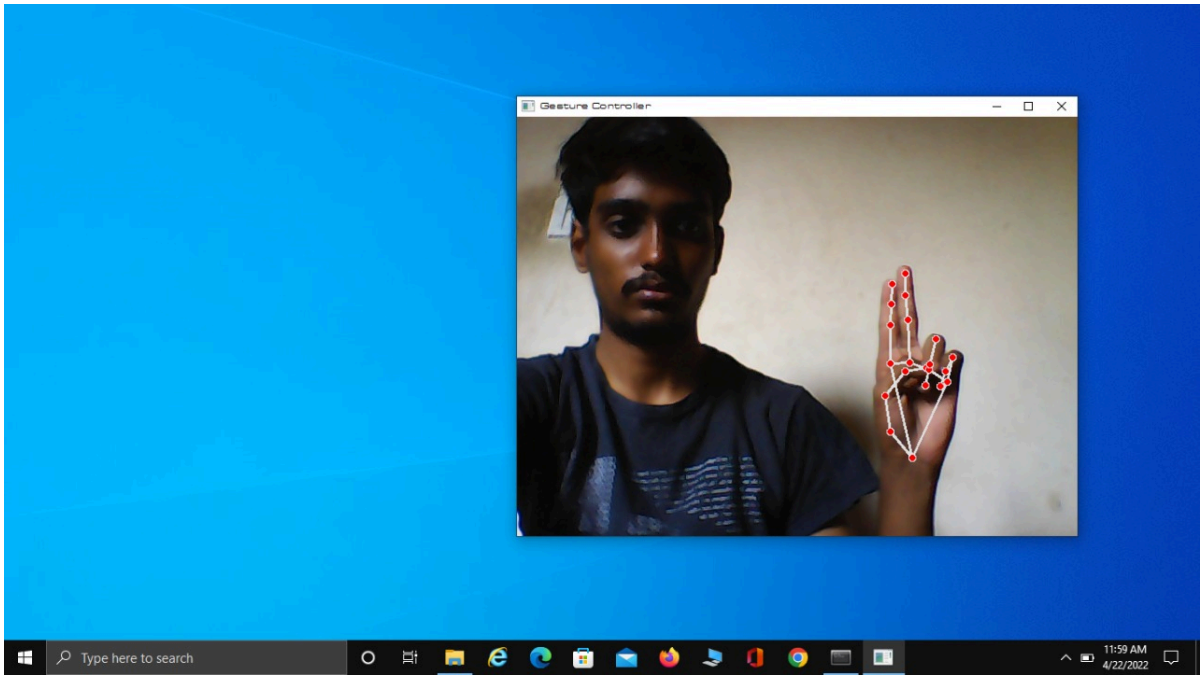

FIGURE 8.5: DOUBLE CLICK

8.6 SCROLLING GESTURE:

Dynamic Gestures for horizontal and vertical scroll. The speed of scroll is proportional to the distance moved by pinch gesture from start point. Vertical and Horizontal scrolls are controlled by vertical and horizontal pinch movements respectively.

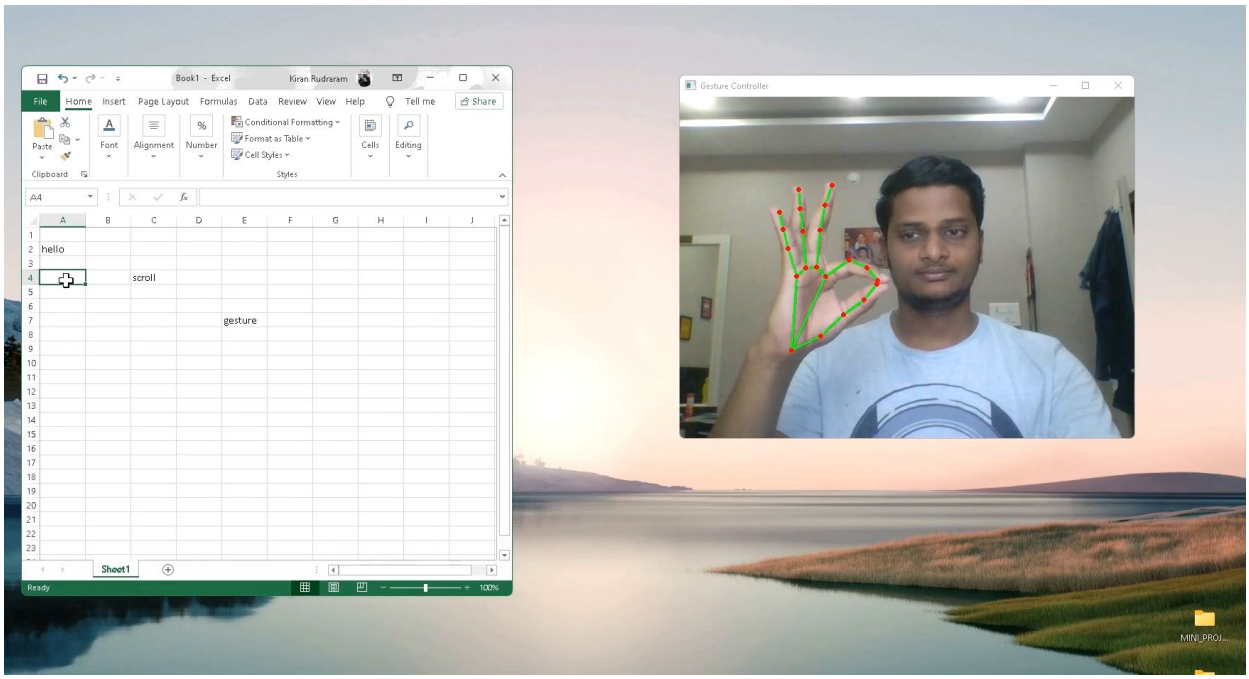

FIGURE 8.6: SCROLLING

8.7 DRAG AND DROP GESTURE:

Gesture for drag and drop functionality. Can be used to move/tranfer files from one directory to other.

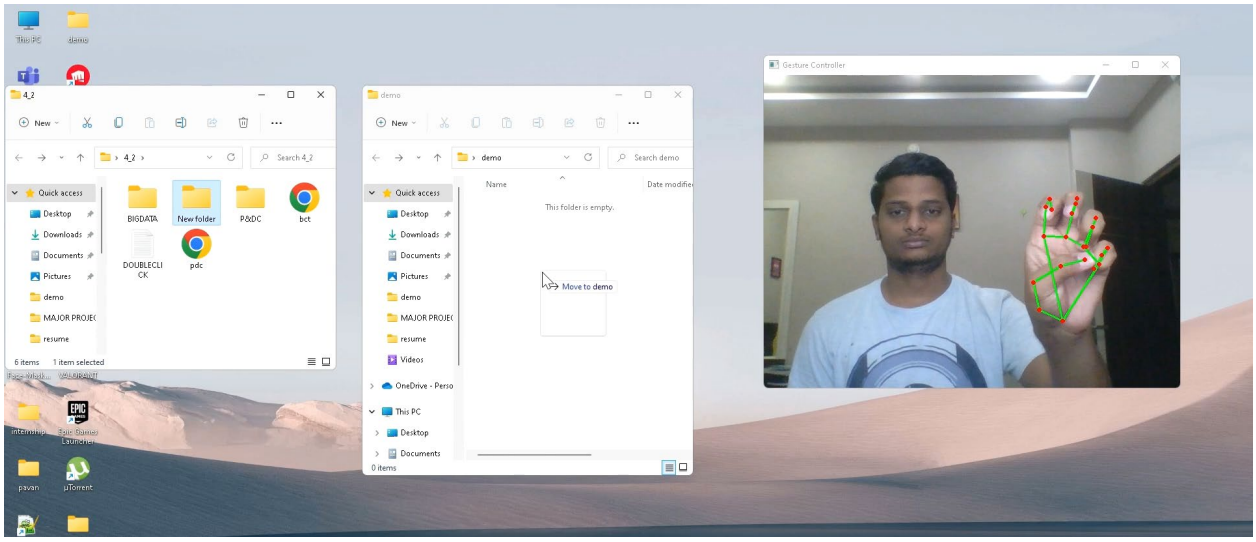

FIGURE 8.7: DRAG AND DROP

8.8 MULTIPLE ITEM SELECTION GESTURE:

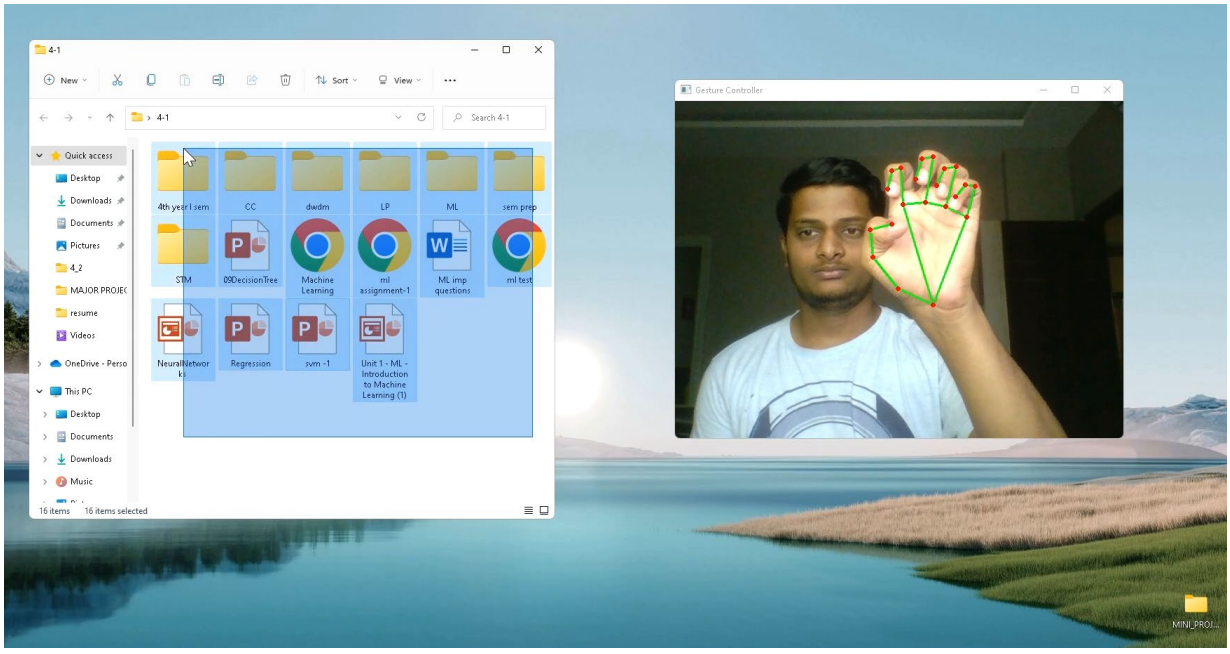

FIGURE 8.8: MULTIPLE ITEM SELECTION

9. CONCLUSION & FUTURE SCOPE

9.1 CONCLUSION:

Gesture recognition gives the best interaction between human and machine. Gesture recognition is also important for developing alternative human computer interaction modalities. It enables human to interface with machine in a more natural way. Gesture recognition can be used for many applications like sign language recognition for deaf and dumb people, robot control etc. This technology has wide applications in the fields of augmented reality, computer graphics, computer gaming, prosthetics, and biomedical instrumentation. This technology can be used to help patients who don't have control of their limbs. In case of computer graphics and gaming this technology has been applied in modern gaming consoles to create interactive games where a person's motions are tracked and interpreted as commands.

9.2 FUTURE SCOPE:

The major extension to this work can be done to make system able to work at much complex background and compatible with different light conditions. It can be made as an effective user interface and which can include all mouse functionalities. And also, it would be ideal to research into advanced mathematical materials for image processing and investigate on different hardware solutions that would result in more accurate hand detections.

10. BIBLIOGRAPHY

10.1 WEBSITES:

- ✓ https://learnopencv.com/
- ✓ https://pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/
- ✓ https://www.geeksforgeeks.org/opencv-python-tutorial/
- ✓ https://learnopencv.com/introduction-to-mediapipe/
- ✓ https://www.geeksforgeeks.org/face-and-hand-landmarks-detection-using-python-mediapipe-opencv/
- ✓ https://www.youtube.com/watch?v=kdLM6AOd2vc&list=PLS1QulWo1RIa7D1O6skqDQ-JZ1GGHKK-K

10.2 REFERENCES:

[1] Abhik Banerjee, Abhirup Ghosh, Koustuvmoni Bharadwaj, “Mouse Control using a Web Camera based on Colour Detection”, IJCTT, vol.9, Mar 2014.

[2] Angel, Neethu.P. S, “Real Time Static & Dynamic Hand Gesture Recognition”, International Journal of Scientific & Engineering Research Volume 4, Issue3, March-2013.

[3] Q. Y. Zhang, F. Chen and X. W. Liu, “Hand Gesture Detection and Segmentation Based on Difference Background Image with Complex Background,” Proceedings of the 2008 International Conference on Embedded Software and Systems, Sichuan, 29-31 July 2008, pp. 338- 343.

[4] J. S. Sonkusare, N. B. Chopade, R. Sor, and S. L. Tade, "A Review on Hand Gesture Recognition System," 2015 Int. Conf Comput. Commun. Control Autom., pp. 790-794,2015.

[5] M. Panwar and P. Singh Mehra, "Hand gesture recognition for human computer interaction," IEEE Int. Conf Image In! Process., 2011 no. Iciip, pp. 1-7.