NEIL GOGTE INSTITUTE OF TECHNOLOGY

(A Unit of Keshav Memorial Technical Education (KMTES)
(Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad).

A

MAJOR PROJECT REPORT

on

GESTURE BRUSH

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

(2023)

VAISHNAVI DAGWALE	:	245319733054
GANUGU SRI VIKAS	:	245319733021
SOHAM CHOUSALKAR	:	245319733051

Under the Guidance

of

Dr. P VENKATESWARLU

(Associate Professor)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NEIL GOGTE INSTITUTE OF TECHNOLOGY
Kachawanisingaram Village, Hyderabad, Telangana – 500058.

May 2023

NEIL GOGTE INSTITUTE OF TECHNOLOGY

(A Unit of Keshav Memorial Technical Education (KMTES)
(Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad).

CERTIFICATE

This is to certify that the Project Report Titled **"Gesture Brush"** is being submitted by **Vaishnavi Dagwale (245319733054), Ganugu Sri Vikas (245319733021)** & **Soham Chousalkar (245319733051),** in **B.E. IV – II Semester** of **Computer Science and 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 **H.O.D.**

External Examiner Signature:

External Viva Voce Date:

NEIL GOGTE INSTITUTE OF TECHNOLOGY
(A Unit of Keshav Memorial Technical Education (KMTES)
(Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad).

DECLARATION

We hereby declare that the Major Project Report entitled, **"Gesture Brush"** submitted for the B.E. degree is entirely our work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree.

Date: _____

By

1. **Vaishnavi Dagwale** **(Signature)**
   245319733054

2. **Ganugu Sri Vikas** **(Signature)**
   245319733021

3. **Soham Chousalkar** **(Signature)**
   245319733051

ACKNOWLEDGEMENT

We are happy to express our deep sense of gratitude to the Principal of the college **Prof. R. Shyam Sunder, Professor, Neil Gogte Institute of Technology,** for having provided us with adequate facilities to pursue our project.

We would like to thank **Dr. K. V. Rangarao, Professor and Head, Dr. B. Krishna, Project Coordinator,** Department of Computer Science and Engineering, Neil Gogte Institute of Technology, for having provided the freedom to use all the facilities available in the department, especially the laboratories and the library.

We are very grateful to our project guide **"P. Venkateshwarlu "**, Associate Professor, Department of Computer Science and Engineering, Neil Gogte Institute of Technology, for his extensive patience and guidance throughout our project work.

We sincerely thank our seniors and all the teaching and non-teaching staff of the Department of Computer Science and Information Technology for their timely suggestions, healthy criticism and motivation during the course of this work.

We would also like to thank classmates for always being there whenever we needed help or moral support With great respect and obedience. We thank our parents, sisters and brother who were the backbone behind our deeds.

Finally, we express our immense gratitude with pleasure to the other individuals who have either directly or indirectly contributed to our need at right time for the development and success of this work.

ABSTRACT

Gesture Brush is a Python-based platform that enables users to draw on a canvas using hand gestures. The current system relies on hardware components such as a mouse, stylus, and touchpads. This can be limiting for artists who want to explore new ways of expression and may not provide the accuracy and precision required for complex artworks.

To address these issues, we propose a new approach that leverages computer vision and deep learning techniques to track hand gestures accurately and precisely. Our solution eliminates the need for hardware components, allowing users to draw directly on the canvas using natural hand movements. Additionally, our approach recognizes complex gestures and translates them into corresponding brush strokes, providing a more intuitive and fluid drawing experience.

The proposed system requires a camera to capture the user's hand movements and a computer to process the data and generate the corresponding brush strokes. We use OpenCV technology for augmented reality and color tracking and detection to produce a visual representation of the user's movements. Our solution will be implemented using Python, making it accessible to a broad community of artists and developers.

LIST OF FIGURES

TABLE OF CONTENTS

1 . INTRODUCTION

Writing in air has been one of the most fascinating and challenging research areas in the field of image processing and pattern recognition in recent years. It contributes immensely to the advancement of an automation process and can improve the interface between man and machine in numerous applications. Several research works have been focusing on new techniques and methods that would reduce the processing time while providing higher recognition accuracy.

Object tracking is considered as an important task within the field of Computer Vision. The invention of faster computers, availability of inexpensive and good quality video cameras and demands of automated video analysis has given popularity to object tracking techniques. Generally, video analysis procedure has three major steps: firstly, detecting the object, secondly tracking its movement from frame to frame and lastly analyzing the behavior of that object. For object tracking, four different issues are taken into account; selection of suitable object representation, feature selection for tracking, object detection and object tracking. In the real world, Object tracking algorithms are the primary part of different applications such as: automatic surveillance, video indexing and vehicle navigation etc.

The project takes advantage of this gap and focuses on developing a motion-to-text converter that can potentially serve as software for intelligent wearable devices for writing from the air. This project is a reporter of occasional gestures. It will use computer vision to trace the path of the finger. The generated text can also be used for various purposes, such as sending messages, emails, etc. It will be a powerful means of communication for the deaf. It is an effective communication method that reduces mobile and laptop usage by eliminating the need to write.

1.1 MOTIVATION

Considering the rise of need for comfort and affordability in each and every field, the purpose of eliminating things comes into picture.

We were motivated by the replacement of fine arts by digital art which involved digitalizing the tools involved, but still there was hardware involved such as a mouse, stylus, drawing tablet.

There was a potential of removing them as well and using the camera instead considering the growth in image recognition technology

1.2 PROBLEM STATEMENT

Creating an application that will enable the user to create digital drawings using hand sign detection on any pc with access to a webcam and numpy, mediapipe, OpenCV installed in it.

1.3 SOLUTION

Using OpenCV, Numpy and mediapipe we create a .ipynb file in which we first import all the required libraries and dependencies for running the code.

1.4 OBJECTIVE

To make it possible for people to digitally draw while only using their hands, this gives them a more natural touch to whatever they are planning to make and also eliminates the involvement of certain hardware devices such as mouse, stylus, touchpad etc.

1.5 LIMITATIONS

- Mediapipe only supports till python 3.10
- It is yet to conquer Three Dimensional art, for now we can only draw in Two Dimensions.

2. LITERATURE SURVEY

- Robust Hand Recognition with Kinect Sensor:

In *Saira Beg, M. Fahad Khan and Faisal Baig, "Text Writing in Air," Journal of Information Display Volume 14, Issue 4, 2013*, the system proposed used the depth and colour information from the Kinect sensor to detect the hand shape. As for gesture recognition, even with the Kinect sensor. It is still a very challenging problem. The resolution of this Kinect sensor is only 640×480. It works well to track a large object, e.g., the human body. But following a tiny thing like a finger is complex.

- LED fitted finger movements:

Authors in *Alper Yilmaz, Omar Javed, Mubarak Shah, "Object Tracking: A Survey", ACM Computer Survey. Vol. 38, Issue. 4, Article 13, Pp. 1-45, 2006* suggested a method in which an LED is mounted on the user's finger, and the web camera is used to track the finger. The character drawn is compared with that present in the database. It returns the alphabet that matches the pattern drawn. It requires a red-coloured LED pointed light source is attached to the finger. Also, it is assumed that there is no red-coloured object other than the LED light within the web camera's focus.

- Augmented Desk Interface:

In *Yuan-Hsiang Chang, Chen-Ming Chang, "Automatic Hand-Pose Trajectory Tracking System Using Video Sequences", INTECH, pp. 132- 152, Croatia, 2010* Augmented segmented desk interface approach for interaction was proposed. This system makes use and a video projector and charge-coupled device (CCD) camera so that using the fingertip; users can operate desktop applications. In this system, each hand performs different tasks. The left hand is used to select radial menus, whereas the right hand is used for selecting objects to be manipulated. It achieves this by using an infrared camera. Determining the fingertip is computationally expensive, so this system defines search windows for fingertips.

3. ANALYSIS AND DESIGN

3.1 EXISTING SYSTEM

Existing system either involve the regular old default drawing using stylus, mouse or touchpad. Sometimes they also use normal drawings, take pictures of them and digitalize them, this method enables them to draw desired content.

There are many other projects out there which use hand sign recognition for virtual drawing, but they too involve some or the other accessory such as holding a pen, or using a colored tape on your fingers.

3.1.1 DISADVANTAGES OF EXISTING SYSTEMS

Existing systems are the same as using conventional digital drawing methods as they too involve keeping the hand busy by making it hold something.

This limits the scenarios where one can use this method to draw.

3.2 PROPOSED SYSTEM

The proposed system involves no such additional hardware but only a camera and strong enough system one's working on.

Here, mere hand gestures are enough for the machine or program to recognise what the user is trying to do and get done.

3.2.1 ADVANTAGES OF PROPOSED SYSTEM

Advantages of our projects involve being able to draw by using mere hands, a decent pc and a working camera.

It eliminates the need of using additional hardware such as stylus, mouse.

3.3 FEASIBILITY ANALYSIS

3.3.1 Economical Feasibility:

- Economical feasibility assesses the financial viability of implementing the gesture-based drawing project. It involves analyzing the costs associated with the development, deployment, and maintenance of the system, as well as the potential benefits and returns on investment. Here are four key aspects of the economical feasibility of this project:

- Cost Analysis: Developing a gesture-based drawing system requires investment in hardware devices such as cameras or sensors, software development, and user interface design. Additionally, ongoing costs for server hosting, maintenance, and potential upgrades need to be considered. Conducting a comprehensive cost analysis will help determine the financial resources required for the project.

- Cost-Benefit Ratio: Assessing the cost-benefit ratio is essential to evaluate the potential returns on investment. The project's benefits include increased productivity, enhanced user experience, and improved accessibility. By comparing the projected benefits with the estimated costs, stakeholders can determine whether the project is economically feasible.

- Revenue Generation: The gesture-based drawing system can potentially generate revenue through various means. For example, it could be marketed as a commercial product or integrated into existing software applications with licensing or subscription models. Evaluating potential revenue streams is crucial for determining the project's long-term sustainability and profitability.

- Market Analysis: Understanding the market demand and competition is crucial for the economic feasibility of the project. Analyzing the target audience, their needs, and existing solutions in the market will help identify potential opportunities and challenges. By conducting market research, stakeholders can make informed decisions about pricing, positioning, and marketing strategies to ensure a competitive advantage.

3.3.2 Technical Feasibility:

- Technical feasibility evaluates the practicality and compatibility of implementing the gesture-based drawing project from a technological perspective. It involves assessing the availability of necessary hardware, software, and technical expertise. Here are four key considerations for the technical feasibility of this project:

- Hardware Requirements: The gesture-based drawing system relies on hardware components such as cameras or sensors capable of capturing and processing hand movements accurately. Assessing the availability and compatibility of such hardware is crucial. Compatibility with different platforms (e.g., PCs, tablets, smartphones) should also be considered for broader usability.

- Software Development: Developing the software components for gesture recognition, image processing, and user interface requires expertise in computer vision, machine learning, and software engineering. Evaluating the availability of skilled developers, relevant libraries, and frameworks is essential for successful implementation.

- Performance and Scalability: The system should be able to handle real-time hand tracking and rendering of drawings without significant latency or performance issues. Scalability considerations should be made to ensure that the system can accommodate a growing number of users and handle increased processing demands.

- Integration and Compatibility: Assessing the compatibility of the gesture-based drawing system with existing software applications, operating systems, or web browsers is vital. Compatibility with popular platforms ensures wider adoption and usability, enhancing the project's technical feasibility.

3.3.3 Social Feasibility:

- Social feasibility examines the impact and acceptance of the gesture-based drawing system within society. It assesses the benefits, potential risks, and social implications of implementing such a system. Here are four key aspects to consider for the social feasibility of this project:

- User Acceptance: Understanding user preferences, needs, and expectations is crucial for social feasibility. Conducting user surveys, interviews, or usability testing sessions can provide insights into user acceptance, satisfaction, and usability. Addressing user concerns and incorporating feedback during the development process enhances the social feasibility of the project.

- Accessibility and Inclusivity: Ensuring that the gesture-based drawing system is accessible to individuals with diverse abilities and disabilities is essential. Considerations such as providing alternative input methods for users with mobility impairments or incorporating assistive technologies will contribute to social inclusivity.

- Learning Curve and Training: Assessing the learning curve required

4 . SYSTEM REQUIREMENT SPECIFICATIONS

4.1 SOFTWARE SYSTEM REQUIREMENTS

- Operating System : Windows 10
- Coding Language : Python 3.x
- Framework : Mediapipe

4.2 HARDWARE SYSTEM REQUIREMENTS

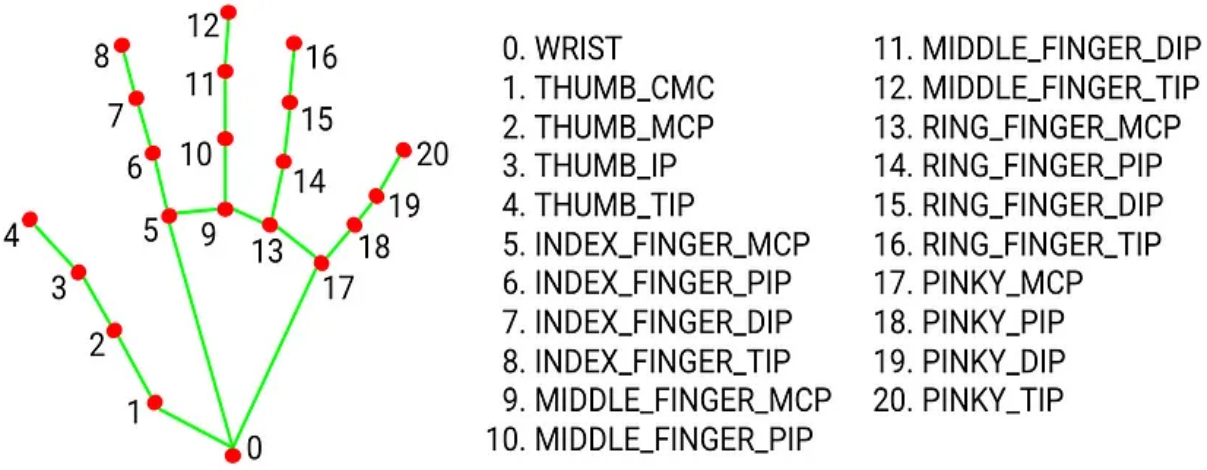
- System : 64 bit OS
- Hard Disk: 1 TB
- RAM : 8 GB.

4.3 SOFTWARE TECHNOLOGIES

This project is developed in python using Mediapipe, OpenCV, NumPy libraries.

4.3.1 MEDIAPIPE 0.8.11

MediaPipe is a customizable machine learning solutions framework developed by Google. It is an open-source and cross-platform framework, and it is very lightweight. MediaPipe comes with some pre-trained ML solutions such as face detection, pose estimation, hand recognition, object detection, etc.



Blazepalm model's palm markings and key points

It is a Framework for building machine learning pipelines for processing time-series data like video, audio, etc. This cross-platform Framework works on Desktop/Server, Android, iOS, and embedded devices like Raspberry Pi and Jetson Nano.

BlazeHand is a machine learning model that detects key points of the hand. Since it can detect detailed hand movements, it can be applied to gesture recognition.

4.3.2 OPENCV 4.6.0

OpenCV is a programming library/package that has been created especially for allowing programmers to enter the world of Computer Vision. The primary developer of the OpenCV package is Intel Corporation.OpenCV stands for Open-Source Computer Vision (Library).OpenCV can be used to process images and videos to identify objects, faces or even the handwriting of the human.

4.3.3 NUMPY 1.21.4

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

5. SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE DESIGN

System Architectural Design:

The system architectural design of the gesture-based drawing project encompasses the overall structure and components required to enable the interaction between the user, the gesture recognition system, and the drawing application. It involves defining the main modules, their interactions, and the data flow within the system. Here is an extensive overview of the system architecture:

User Interface:

The user interface serves as the bridge between the user and the system. It provides a visual representation of the drawing canvas and allows users to interact with the application. The user interface includes elements such as the drawing area, color palette, tool selection, and feedback mechanisms to provide a seamless and intuitive experience.

Gesture Recognition Module:

At the core of the system is the gesture recognition module, responsible for capturing and interpreting hand movements. It utilizes computer vision and machine learning techniques to analyze the video feed from the camera or sensor. This module identifies specific hand gestures, tracks the movement of the user's hand, and translates them into actionable commands for the drawing application.

Drawing Application:

The drawing application module handles the rendering and manipulation of graphical elements based on the user's hand gestures. It provides functionalities such as drawing lines, shapes, changing colors, and erasing. The module receives input from the gesture recognition module and updates the canvas accordingly.

Data Flow:

The data flow within the system follows a well-defined process. The camera or sensor captures video frames, which are then passed to the gesture recognition module for analysis. The module extracts relevant hand features and recognizes specific gestures. The recognized gestures are then transmitted to the drawing application module, which updates the canvas based on the received commands.

Integration:

To ensure seamless integration and communication between the different modules, appropriate interfaces and protocols are established. For example, the gesture recognition module sends recognized gestures to the drawing application module using a predefined data format or API. This integration enables real-time synchronization between hand movements and the resulting graphical output.

Performance and Optimization:

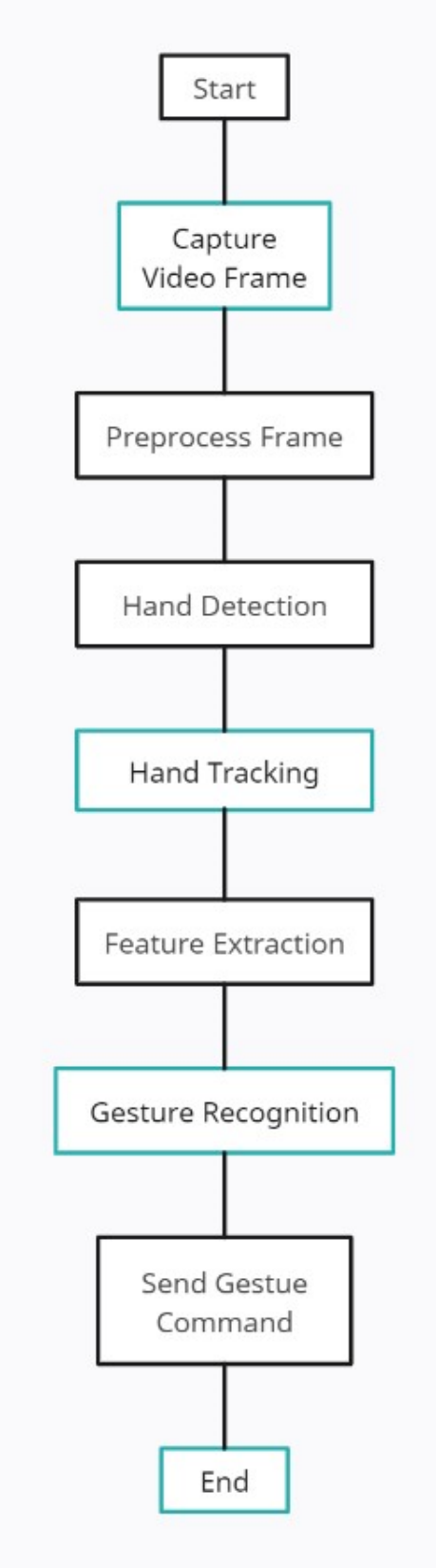
To ensure optimal system performance, various optimization techniques can be employed. This includes implementing efficient algorithms for hand tracking and gesture recognition, minimizing latency, and optimizing the rendering process. Performance monitoring and profiling tools can be used to identify bottlenecks and improve the overall responsiveness of the system.

Flow Charts:

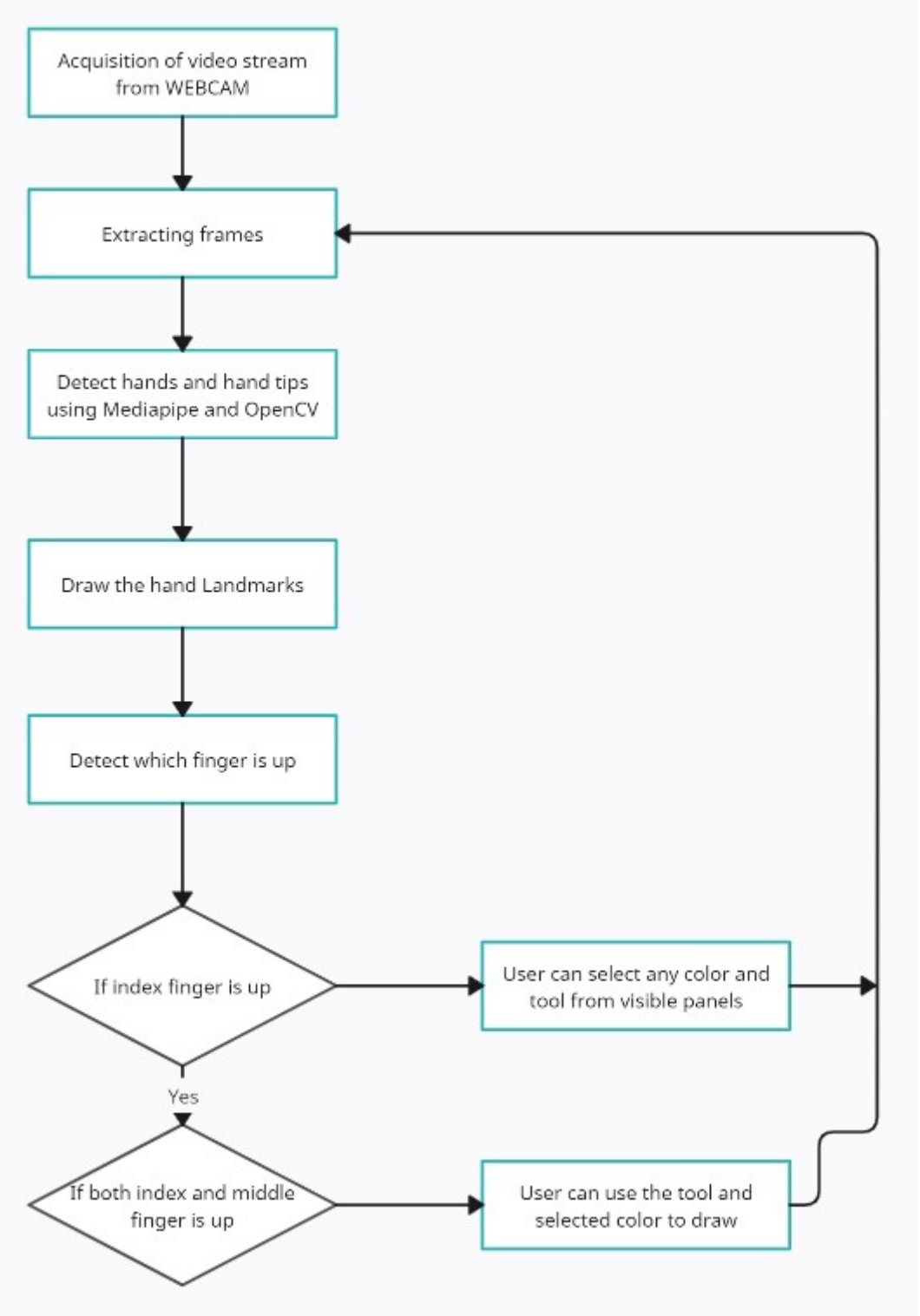
Flow charts provide a visual representation of the system's logical flow and the sequence of actions performed during different stages. Here are two flow charts illustrating the key processes in the gesture-based drawing system:

Gesture Recognition Flow Chart:

This flow chart outlines the steps involved in the gesture recognition process:



5.2 DATA FLOW DIAGRAM



5.3 UML DIAGRAMS

Below are the UML diagrams for Gesture Brush.

5.3.1 CLASS DIAGRAM

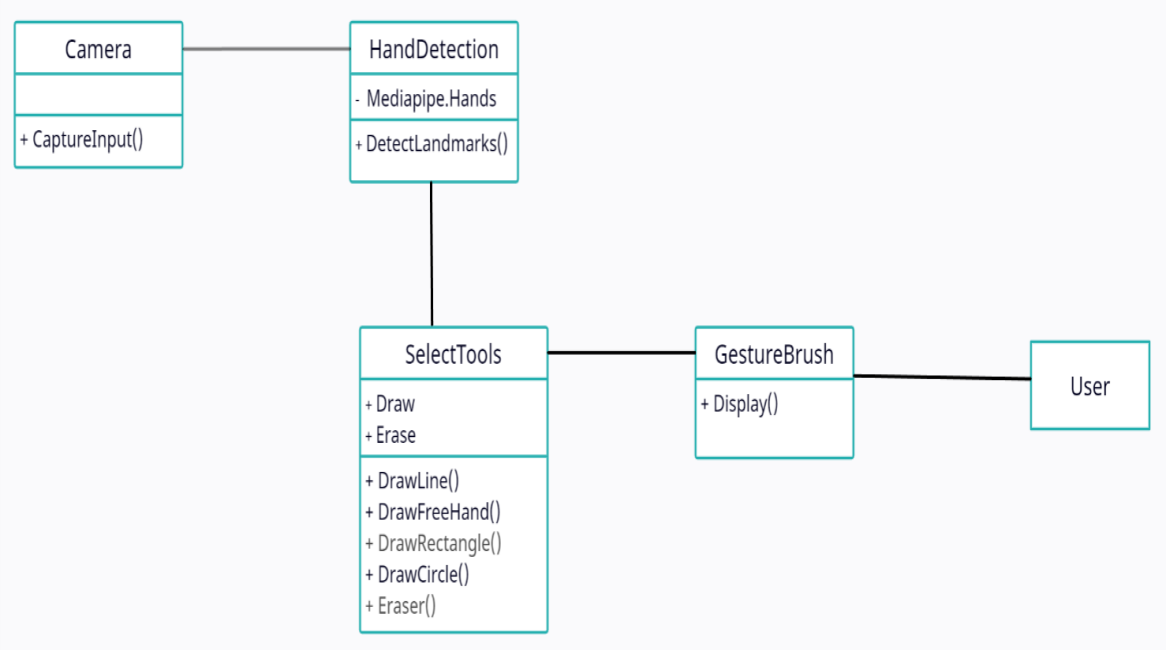


Fig 5.1 Class Diagram for Drawing using Hand Gesture

5.3.2 ACTIVITY DIAGRAM

From the start node the first activity is to accept images from the ROI input frame.Then the image is converted from RGB to Grayscale and resized to the dimension 28x28x1.Then the image is sent to the model where the corresponding label to the english letter is predicted.Then the English letter of the predicted layer is displayed on the screen and the end node is reached.

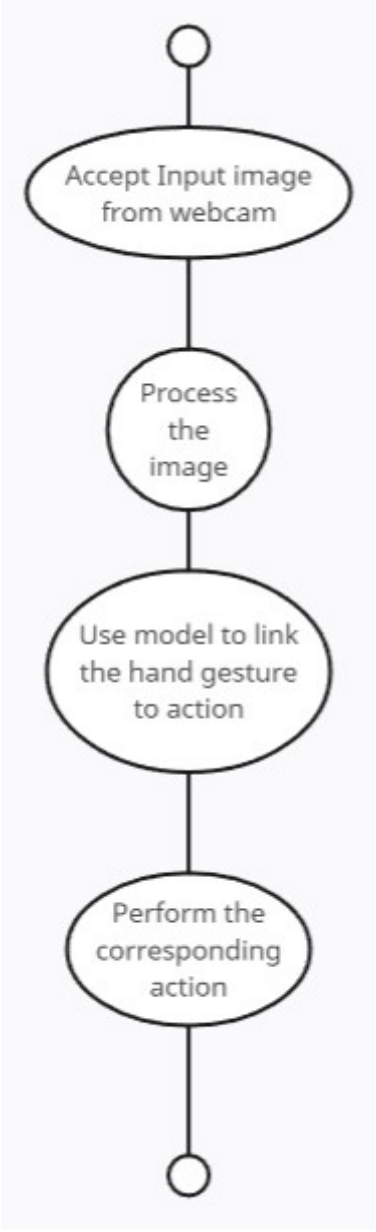


Fig 5.2 Activity Diagram for Hand Gesture Recognition

5.3.3 SEQUENCE DIAGRAM

Webcam will be used to acquire the image to be recognized from the ROI(Region Of Interest). The image is processed by converting it from RGB to Grayscale.The image is then resized to 28x28x1 and sent to the model.The recognition is done by the pre-trained model and the label corresponding to the ASL gesture being signed is returned.

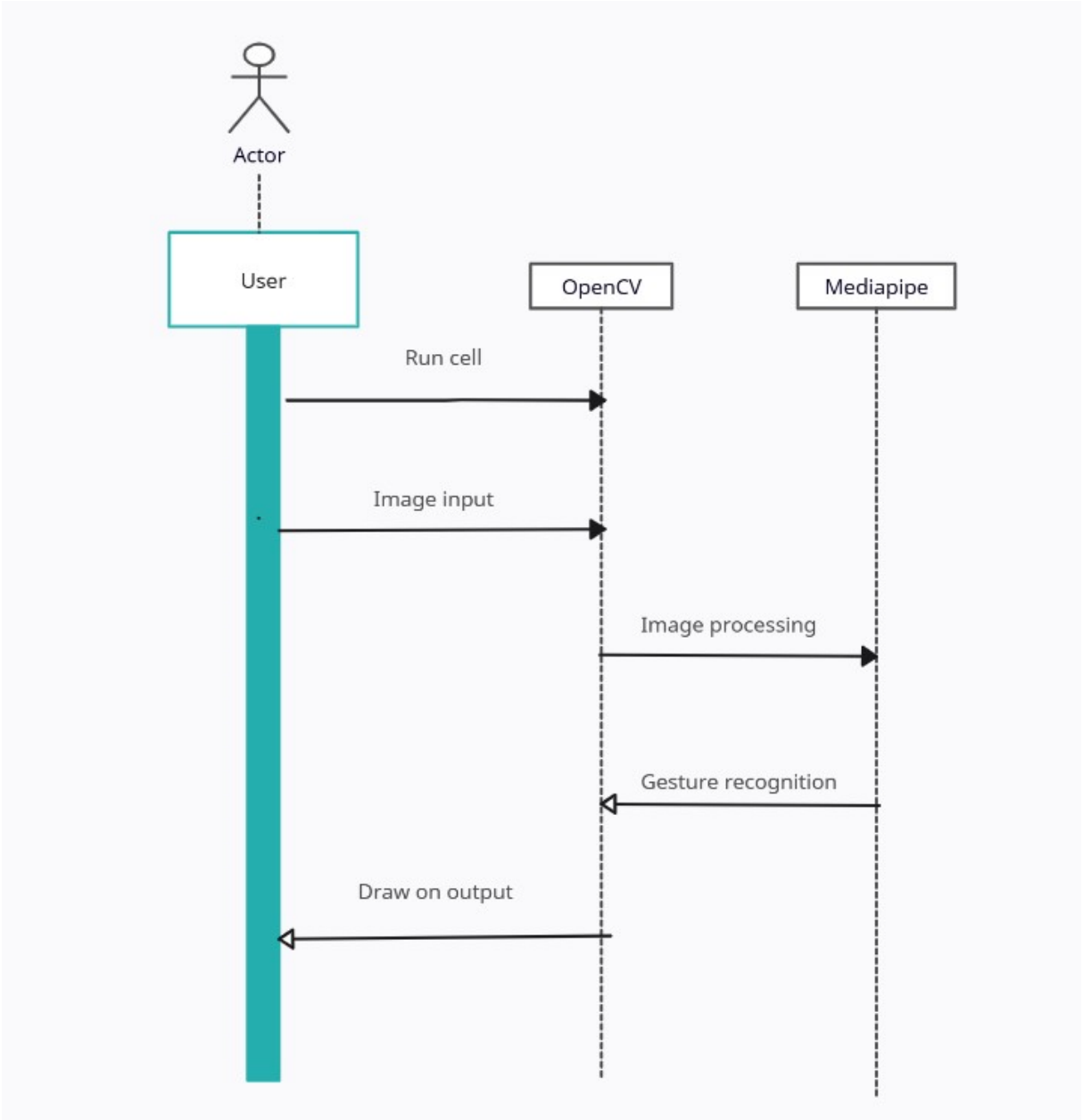


Fig 5.3 Sequence Diagram for Gesture Brush

5.3.4 USE CASE DIAGRAM

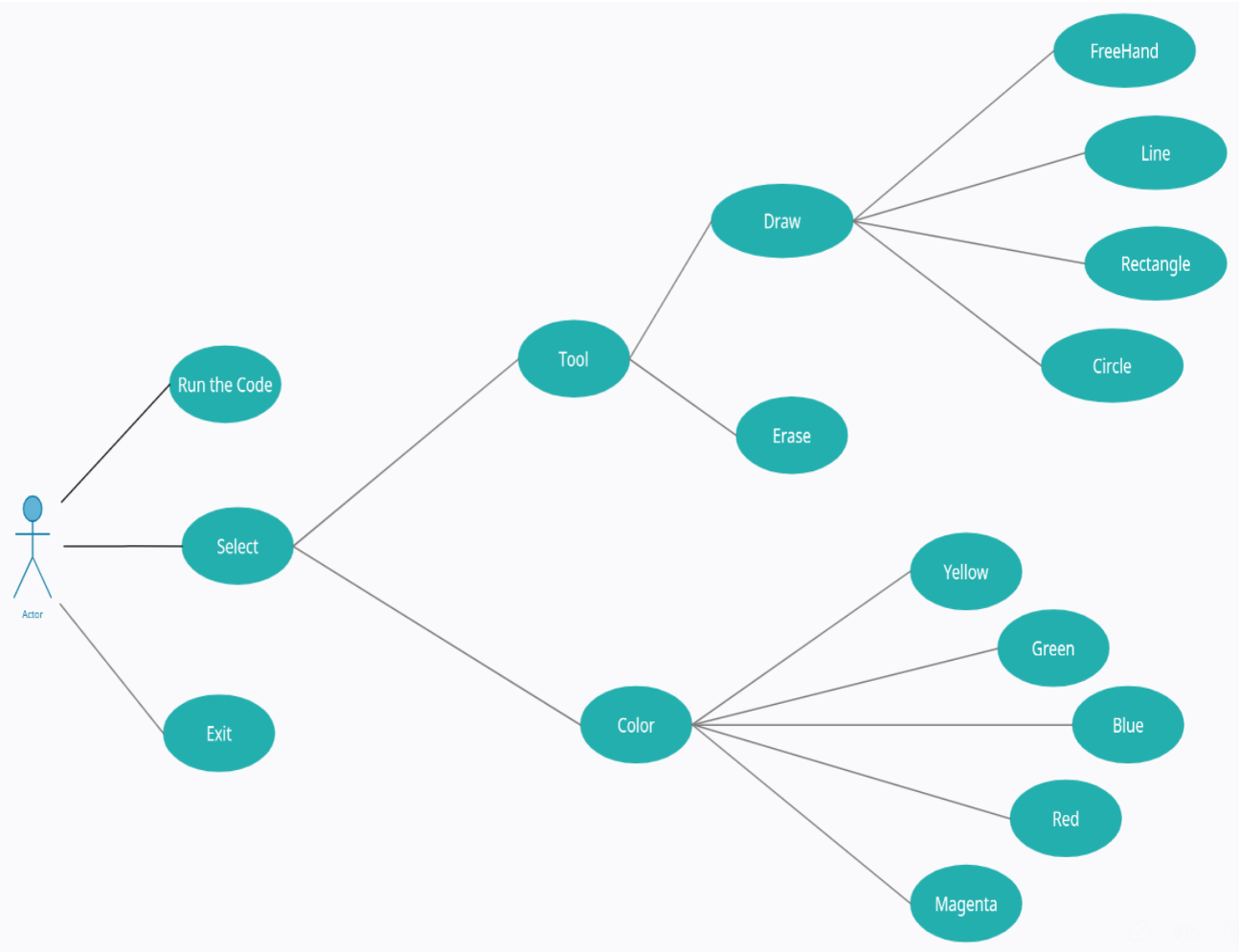


Fig 5.4 Use Case Diagram for Gesture Brush

Participating Actors	User,System
Follow of Events	Start the system webcam Open the Paint Window Select the desired Tools and Colors Begin drawing Witness your drawing
Entry Condition	Run the code
Exit Condition	See your drawing
Quality Requirements	Camera pixels quality, lightening

Table 5.4 UseCase Scenario for Gesture Brush

5.3.5 ENTITY RELATIONSHIP DIAGRAM

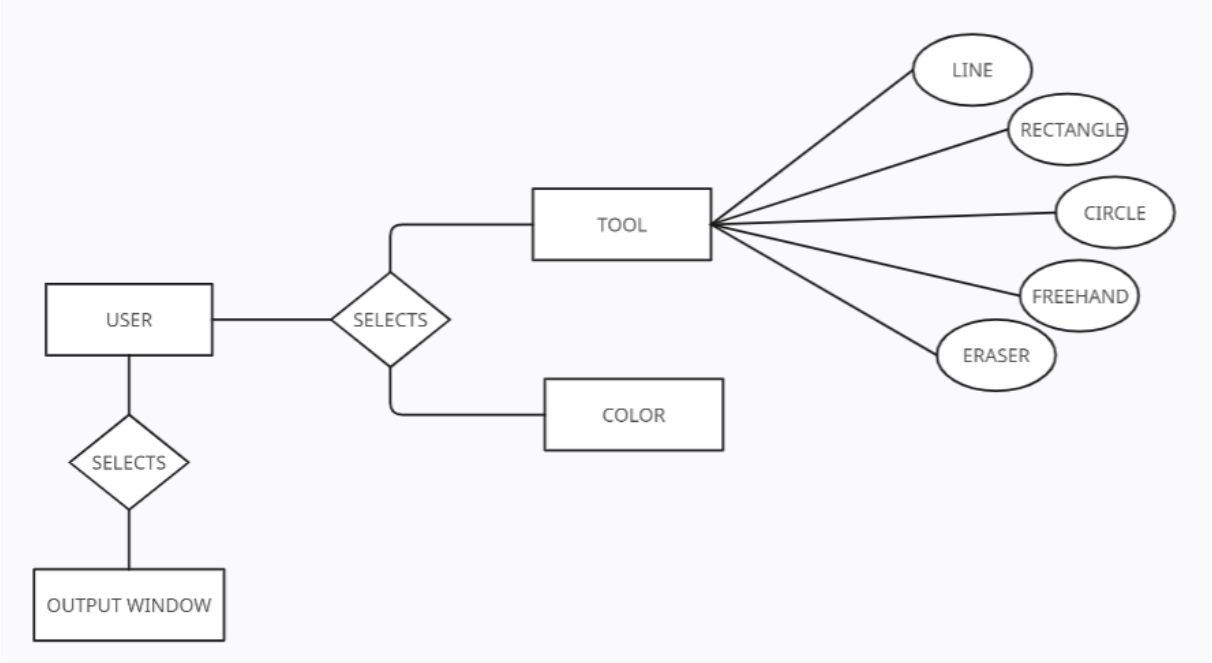


Fig 5.5 ER Diagram of Gesture Brush

6. SYSTEM IMPLEMENTATION

This project aims to create a real-time air canvas which one can easily use.The model is simple and can easily be loaded on any CPU. We implemented our virtual drawing application on python using OpenCV, Mediapipe and numpy libraries. We followed these steps to execute our project plan:

- We installed OpenCV, Mediapipe and Numpy on our computers using pip command.
- We used OpenCV to capture and process the video stream from the webcam.
- We used Mediapipe to detect and track the hand landmarks using a holistic model that combines face, hand, and pose estimation.
- We used OpenCV to map the hand landmarks to the screen coordinates using perspective transformation and calibration techniques.
- We used OpenCV to draw on the screen using drawing functions such as line, circle, and rectangle.
- We used numpy to manipulate arrays and convert colors between different color spaces.
- We used OpenCV to recognize different hand gestures using contour analysis and gesture recognition algorithms.
- We used OpenCV to provide different options for the user to choose the color, thickness, and shape of the drawing.

There are two modes in our project, namely:

- Selection:

  Selection mode is enabled when only the index finger is raised.

  - Select Tool:

    This mode enables the user to select the desired tools namely circle, rectangle, free hand drawing, line or an eraser.

    The selected tool name is displayed on the upper right corner of the window.

- Select Color:

  This mode enables the user to select the color from the available color pane.

  The selected color is flashed upon selection and the selected tool is displayed in the color selected.

- Drawing

  When Index finger and Middle finger are raised, the drawing mode is enabled.

  The drawing mode is determined by the positions of the middle finger.

  User can now draw in the air with the selected tool and color.

  To disable the drawing mode, simply lower your middle finger position.

6.1 PALM DETECTION

BlazeHand is a machine learning model that detects key points of the hand. Since it can detect detailed hand movements, it can be applied to gesture recognition.
BlazeHand consists of two models, BlazePalm and BlazeHand. After detecting the hand position from the input image with BlazePalm, keypoints of the hand are detected from the hand image with BlazeHand.

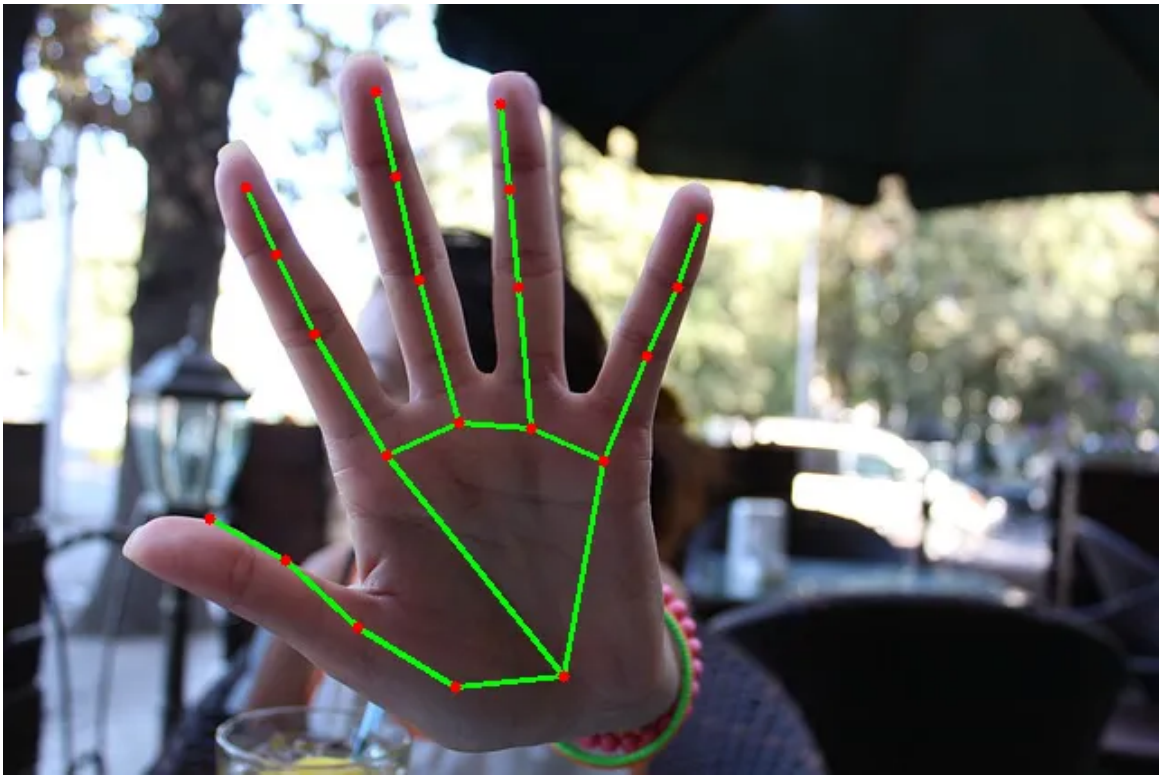


Fig 6.1 Palm detection using BlazePalm algorithm

The hand detection by BlazePalm is very demanding if it is processed every frame, it may also lose track of the hand. Therefore, in the first frame, BlazePalm performs the detection process, and in subsequent frames, it calculates a slightly larger Rectangle (ROI) from the key point of the hand detected by BlazeHand, and applies BlazeHand to that Rectangle to move the Rectangle. This enables fast and robust recognition.

6.2 CNN ARCHITECTURE

The CNN is trained to detect the palm of a hand in an image, and output the location of 21 landmarks on the palm.

The architecture of the CNN used in Mediapipe's palm detection model consists of three main components: the input layer, the convolutional layers, and the output layer.

Input layer: The input layer of the CNN takes an image of size 256x256 pixels as input. This image is first resized to 256x256 pixels and then normalized to have pixel values in the range [0, 1].

Convolutional layers: The convolutional layers of the CNN perform feature extraction by applying a set of learnable filters to the input image. Each filter extracts a specific feature from the input image, such as edges, corners, or texture patterns. The output of each convolutional layer is a set of feature maps, which are passed to the next layer. The palm detection model in Mediapipe uses a total of 15 convolutional layers, with varying filter sizes and numbers of filters.

Output layer: The output layer of the CNN produces the location of the 21 landmarks on the palm of the hand. This is done by applying a set of fully connected layers to the output of the convolutional layers. The output of the final fully connected layer is a set of 63 values, which represent the x, y, and z coordinates of the 21 landmarks.

To train the CNN, the Mediapipe team used a dataset of over 30,000 hand images, annotated with the location of the palm landmarks. The CNN was trained using the cross-entropy loss function and the Adam optimization algorithm.

During inference, the input image is passed through the CNN, and the output of the final layer is decoded to obtain the location of the 21 landmarks on the palm. These landmarks can then be used for further hand tracking and gesture recognition tasks.

Once the network is trained, it is tested on the testing dataset. The model is trained on 20 epochs with a batch size of 128. Fig 5.5.1 shows the images of the predicted ASL Alphabets on the trained CNN model. Excluding J and Z it predicted 20 alphabets accurately. Some errors can be suspected due to similarity in some ASL letters which leads to some faulty predictions. Since we are using the MNIST dataset with adequate images for each letter,most of the

predictions are correct.

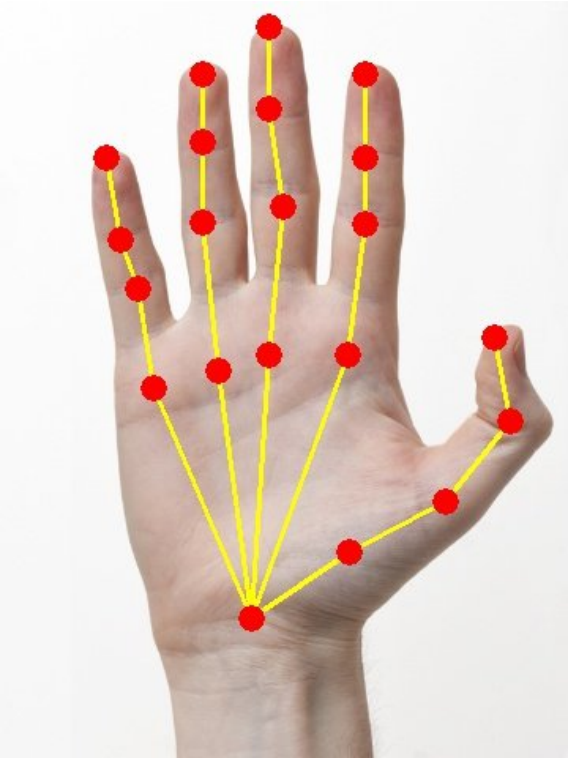


Fig 6.2 Prediction of the CNN model

6.3 SOURCE CODE

#Import required Libraries

```
import mediapipe as mp

import cv2

import numpy as np
```

#Constants

```
curr_tool = "Select Tool"

curr_color = (22, 22, 255) #Default color: bright red

trad = 15 #tools selection radius

crad = 15 #color selection radius

var_inits = False #variables initialised? No

bthickness = 4 #brush thickness

erad = 30 #erase radius

x_prev, y_prev = 0, 0 #initalize variables
```

#Select Tool function

```
def getTool(x):

        if x<200:

                return "Draw"

        elif x<250:
```

```
            return "Line"
        elif x <300:
            return"Rectangle"
        elif x<350:
            return "Circle"
        else:
            return "Erase"
```

#Select Color function

```
def getColor(y):
    if y<100:
        return (89, 222, 255) #Yellow
    elif y<140:
        return (87, 217, 126) #Grass Green
    elif y<180:
        return (203, 194, 0) #Aqua Blue
    elif y<220:
        return (22, 22, 255) #Bright Red
    else:
        return (230, 108, 203) #Magenta
```

#Mode Selection:: Middle finger positions: MIDDLE_FINGER_TIP - MIDDLE_FINGER_PIP

```
def fingers_up(y12, y10):
    if y10>y12:
        return True
    return False
```

#Detecting Hand Initialisation

```
hands = mp.solutions.hands
hand_landmark = hands.Hands(min_detection_confidence=0.6, min_tracking_confidence=0.6, max_num_hands=1)
draw = mp.solutions.drawing_utils
drawColor = mp.solutions.drawing_styles
```

#Canvas Visualisation

```
tools = cv2.imread("tools.png")
colors=cv2.imread("colors.png")
mask = np.zeros((480, 640, 3), np.uint8)
```

#Implementation

```
cap = cv2.VideoCapture(0)
cap.set(3, 480)
```

```
cap.set(4, 640)

while True:

    _, frm = cap.read()

    frm = cv2.flip(frm, 1)

    rgb = cv2.cvtColor(frm, cv2.COLOR_BGR2RGB)

    op = hand_landmark.process(rgb)

    if op.multi_hand_landmarks:

        for i in op.multi_hand_landmarks:

            draw.draw_landmarks(frm, i, hands.HAND_CONNECTIONS,
drawColor.get_default_hand_landmarks_style(),
drawColor.get_default_hand_connections_style())

            x_ind, y_ind = int(i.landmark[8].x*640), int(i.landmark[8].y*480)

            # set color

            if x_ind<50 and y_ind>60 and y_ind<260:

                cv2.circle(frm, (x_ind, y_ind), crad, (0,0,0), 3)


                curr_color = getColor(y_ind)

                frm[:40,55:95] = curr_color

                #print("your current color is set to : ", curr_color)

            # select tool

            if y_ind<50 and x_ind>150 and x_ind<400:

                cv2.circle(frm, (x_ind, y_ind), trad, (0,0,0), 3)
```

```
                curr_tool = getTool(x_ind)
                #print("your current tool is set to : ", curr_tool)
            #Detecting positions 12,10
            y12 = int(i.landmark[12].y*480)
            y10 = int(i.landmark[10].y*480)
            # Freehand Drawing
            if curr_tool == "Draw":
                # select
                if fingers_up(y12, y10):
                    cv2.line(mask, (x_prev, y_prev), (x_ind, y_ind),
curr_color, bthickness)
                    x_prev, y_prev = x_ind, y_ind
                # fix
                else:
                    x_prev = x_ind
                    y_prev = y_ind
            # Drawing Line
            elif curr_tool == "Line":
                # select
                if fingers_up(y12, y10):
                    if not(var_inits):
                        x_ind_old, y_ind_old = x_ind, y_ind
```

```
                    var_inits = True
                cv2.line(frm, (x_ind_old, y_ind_old), (x_ind, y_ind),
curr_color, bthickness)
            # fix
            else:
                if var_inits:
                    cv2.line(mask, (x_ind_old, y_ind_old), (x_ind,
y_ind), curr_color, bthickness)
                    var_inits = False
        # Drawing Rectangle
        elif curr_tool == "Rectangle":
            # select
            if fingers_up(y12, y10):
                if not(var_inits):
                    x_ind_old, y_ind_old = x_ind, y_ind
                    var_inits = True
                cv2.rectangle(frm, (x_ind_old, y_ind_old), (x_ind, y_ind),
curr_color, bthickness)
            # fix
            else:
                if var_inits:
                    cv2.rectangle(mask, (x_ind_old, y_ind_old),
(x_ind, y_ind), curr_color,bthickness)
```

```
                    var_inits = False
            # Drawing Circle
            elif curr_tool == "Circle":
                # select
                if fingers_up(y12, y10):
                    if not(var_inits):
                        x_ind_old, y_ind_old = x_ind, y_ind
                        var_inits = True

cv2.circle(frm,(x_ind_old,y_ind_old),int(((x_ind_old-x_ind)**2+(y_ind_old-y_ind)**2)**0.5)
,curr_color,bthickness)
                # fix
                else:
                    if var_inits:

cv2.circle(mask,(x_ind_old,y_ind_old),int(((x_ind_old-x_ind)**2+(y_ind_old-y_ind)**2)**0.
5),curr_color,bthickness)
                        var_inits = False
            # Erase
            elif curr_tool == "Erase":
                # erasing
                if fingers_up(y12, y10):
                    cv2.circle(frm, (x_ind, y_ind), erad, (0,0,0), -1)
```

```
                                cv2.circle(mask, (x_ind, y_ind), erad, 0, -1)


        #Frame and Mask Integration
        graym=cv2.cvtColor(mask,cv2.COLOR_BGR2GRAY)
        _, inv=cv2.threshold(graym,50,255,cv2.THRESH_BINARY_INV)
        inv=cv2.cvtColor(inv,cv2.COLOR_GRAY2BGR)
        frm=cv2.bitwise_and(frm,inv)
        frm=cv2.bitwise_or(frm,mask)


        frm[:50,150:400] = cv2.addWeighted(tools, 0.7, frm[:50,150:400], 0.3, 0)
        frm[:260,:50]=cv2.addWeighted(colors,0.7,frm[:260,:50],0.3,0)
        cv2.putText(frm, curr_tool, (420,30), cv2.FONT_HERSHEY_TRIPLEX, 1, curr_color,
2)
        cv2.imshow("Gesture Brush", frm)
        cv2.imshow("maskgray",graym)
        cv2.imshow("mask",mask)
        cv2.imshow("inv",inv)


        if cv2.waitKey(1) == 27: #Esc key
                cv2.destroyAllWindows()
                cap.release()
                break
```

7 .SYSTEM TESTING AND VALIDATION

7.1 INTRODUCTION TO TESTING

- The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the
- Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

7.2 UNIT TESTING

- Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .It is done after the completion of an individual unit before integration. This is a structural testing that relies on knowledge of its construction and is invasive.
- Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

7.3 INTEGRATION TESTING

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

7.4 ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

7.5 TEST APPROACH

- There are several testing strategies and approaches that can be used to ensure the accuracy, completeness, and usability of project documentation. Testing strategies may include reviews and walkthroughs, user acceptance testing, automated testing, performance testing, and compliance testing.
- Approaches to testing project documentation may include requirements-based testing, risk-based testing, exploratory testing, regression testing, and compliance testing. The selection of the most appropriate strategy and approach will depend on the specific needs and goals of the project, as well as the available resources and expertise of the testing team.

8 . OUTPUT SCREENSHOTS

Below are the screenshots of the outputs obtained.

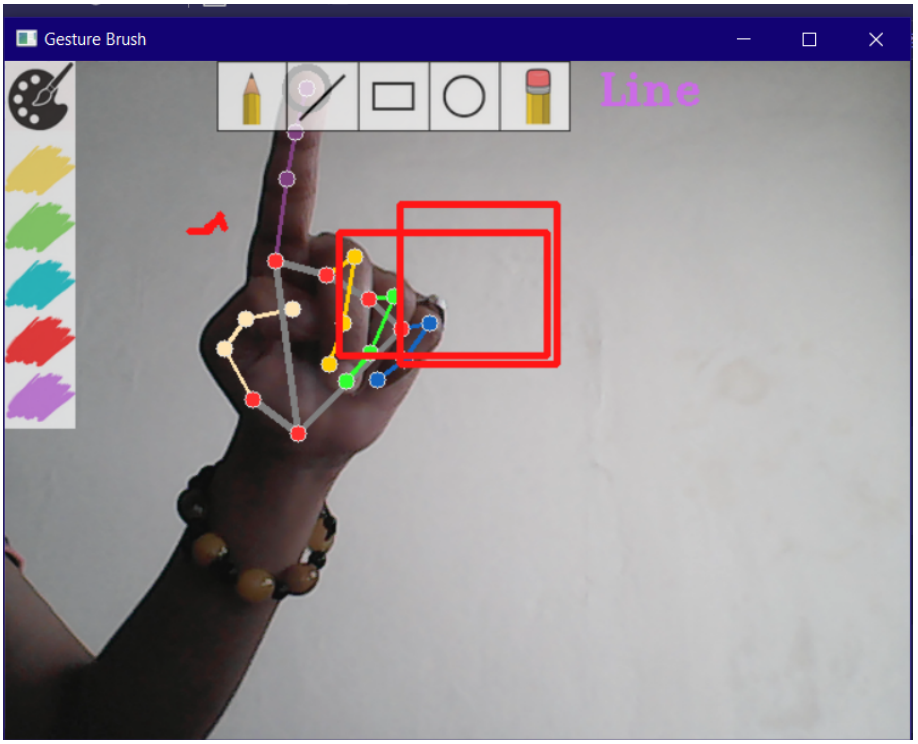


Fig 8.1. Selection mode: Using index finger to select tool

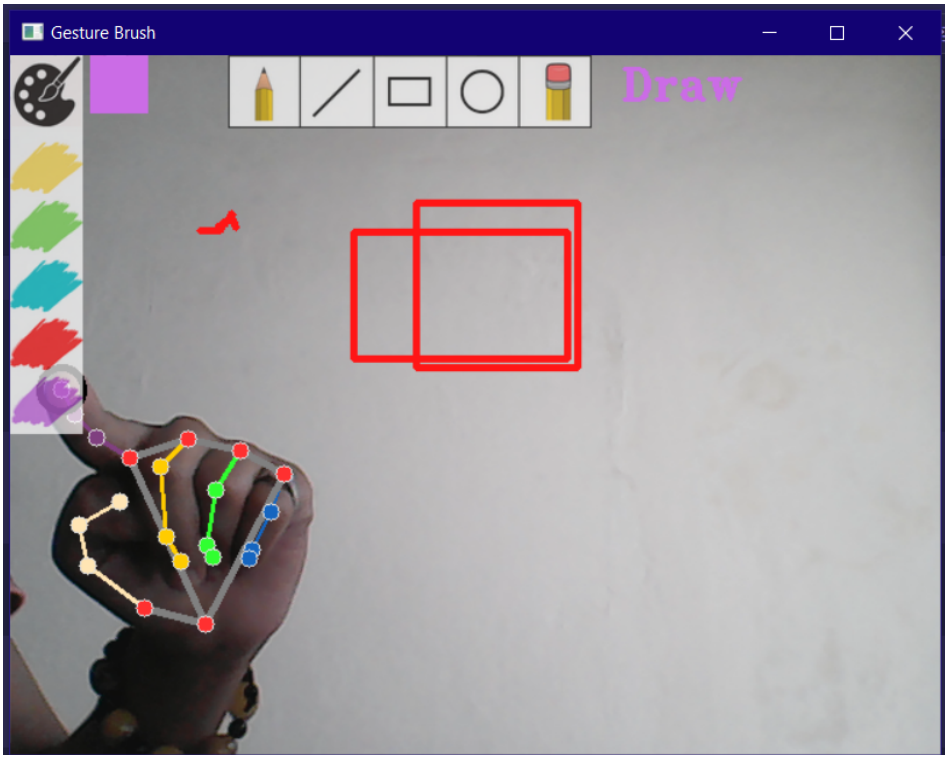


Fig 8.2 Selection mode: Using Index finger to select color

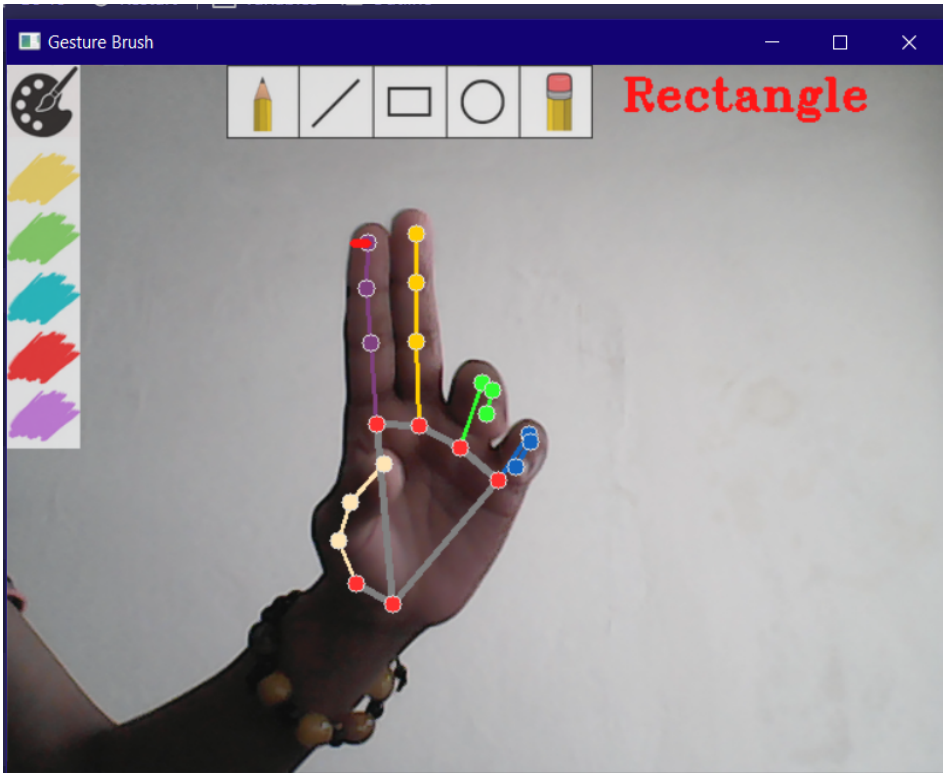


Fig 8.3 Drawing Mode Enabled

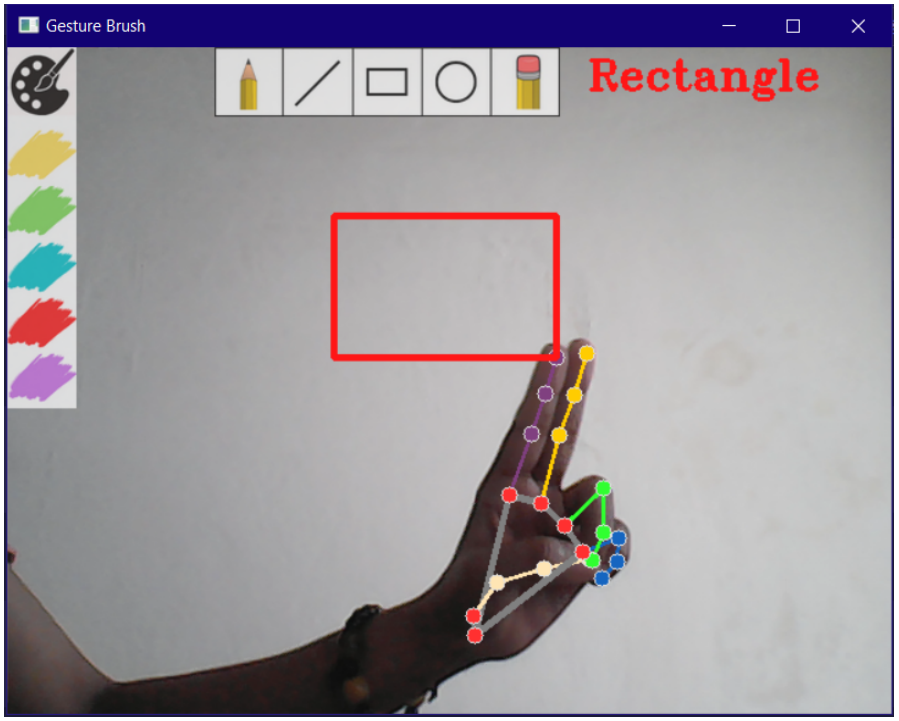


Fig 8.4 Drawing Mode: Drawing using rectangle tool

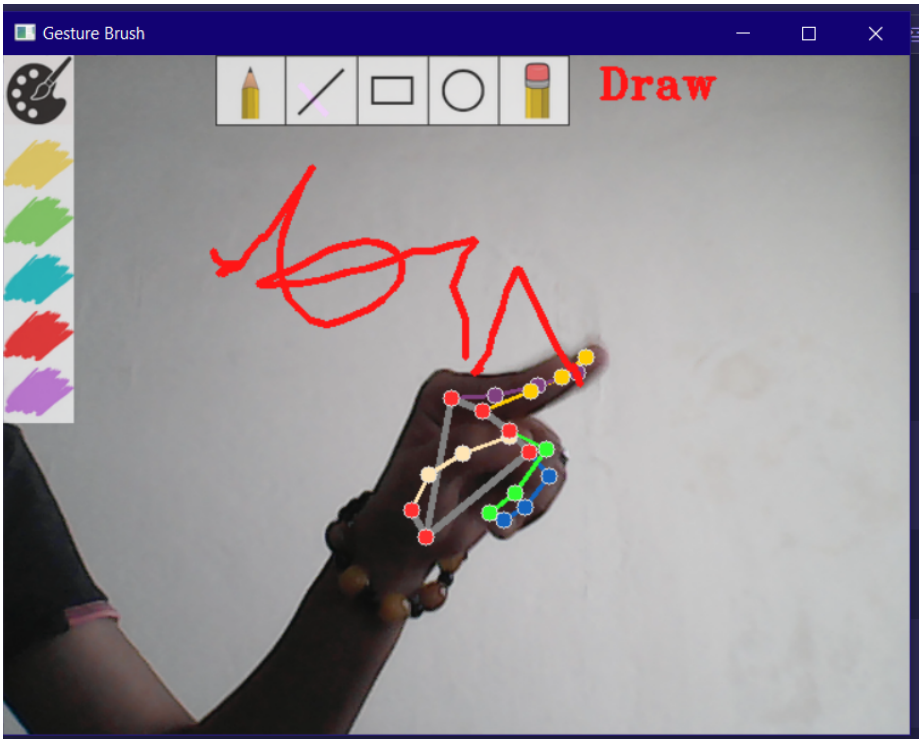


Fig 8.5 Drawing Mode: Drawing using freehand

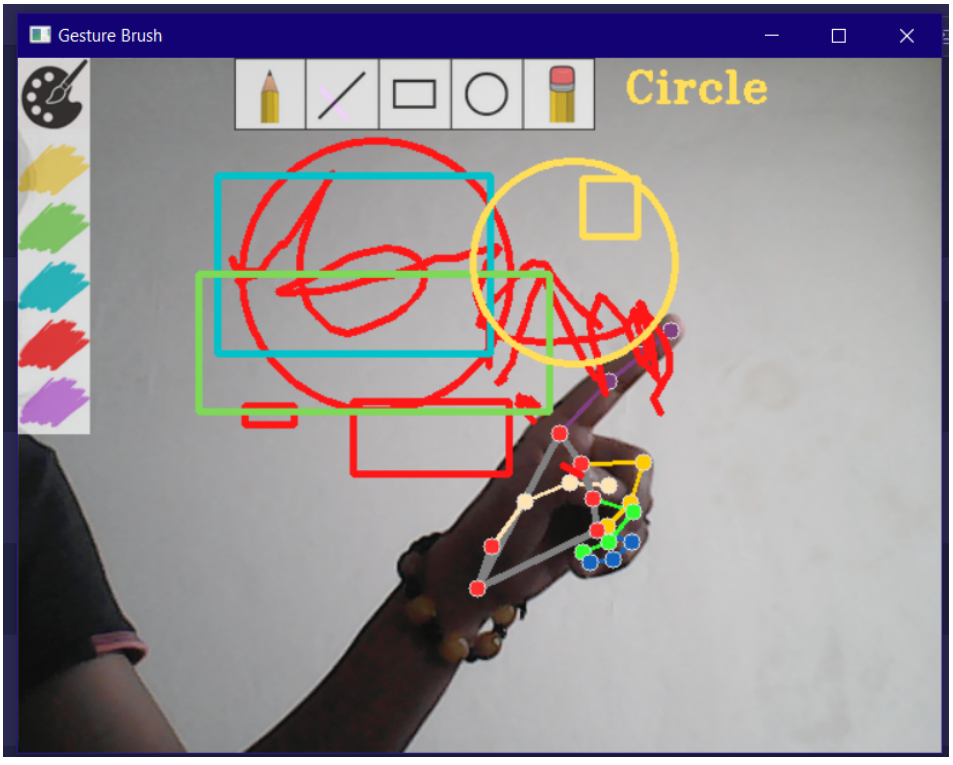


Fig 8.6 Drawing Mode: Drawing using circle tool

Fig 8.7 Drawing Mode: Erasing all the content

9. CONCLUSION AND FUTURE SCOPE

Conclusion

- In this project, we have developed a virtual drawing application on python that uses OpenCV, Mediapipe and numpy. The application enables the user to draw on the screen using their hand gestures, without the need for any physical device. The application leverages the power of OpenCV, Mediapipe and numpy to perform various tasks, such as:
- Detecting and tracking the hand landmarks using Mediapipe's holistic model, which combines face, hand and pose estimation.
- Mapping the hand landmarks to the screen coordinates using OpenCV's perspective transformation and calibration techniques.
- Drawing on the screen using OpenCV's drawing functions, such as line, circle and rectangle.
- Providing different options for the user to choose the color, thickness and shape of the drawing using numpy's array manipulation and OpenCV's color conversion methods.
- Recognizing different hand gestures using Mediapipe's classification model and OpenCV's contour analysis and gesture recognition algorithms.
- The application is user-friendly, interactive and fun to use. It allows the user to express their creativity and imagination through virtual drawing. The application also demonstrates the potential of computer vision and machine learning in creating innovative and engaging applications.

Future Scope

- The virtual drawing application has a lot of potential for further improvement and enhancement. Some of the possible future works are:
- Adding more features and functionalities to the application, such as erasing, undoing, saving and loading the drawings. This would require implementing file handling and data serialization techniques using python's built-in modules or external libraries.
- Improving the accuracy and robustness of the hand gesture recognition, especially in different lighting conditions and backgrounds. This would require fine-tuning the parameters and thresholds of the Mediapipe and OpenCV models, or training custom

models using deep learning frameworks such as TensorFlow or PyTorch.

- Extending the application to support multiple users and collaborative drawing sessions. This would require implementing networking and communication protocols using python's socket or web frameworks such as Flask or Django.
- Integrating the application with other platforms and devices, such as web browsers, mobile phones and tablets. This would require adapting the application to different screen sizes and resolutions, or using cross-platform tools such as Kivy or PyQt.
- Exploring other applications of hand gesture recognition, such as gaming, education and entertainment. This would require designing new interfaces and interactions using hand gestures, or integrating the application with existing applications or platforms such as Unity or Scratch.

10. BIBLIOGRAPHY AND REFERENCES

- https://medium.com/axinc-ai/blazehand-a-machine-learning-model-for-detecting-hand-key-points-c3943b82739a
- https://opencv.org/releases/
- https://pypi.org/project/mediapipe/
- https://numpy.org/
- https://app.creately.com/d/nMksGhDFJXK/edit/s/MTIIKWJEdQ8
- Saira Beg, M. Fahad Khan and Faisal Baig, "Text Writing in Air," Journal of Information Display Volume 14, Issue 4, 2013
- Alper Yilmaz, Omar Javed, Mubarak Shah, "Object Tracking: A Survey", ACM Computer Survey. Vol. 38, Issue. 4, Article 13, Pp. 1-45, 2006
- Yuan-Hsiang Chang, Chen-Ming Chang, "Automatic Hand-Pose Trajectory Tracking System Using Video Sequences", INTECH, pp. 132- 152, Croatia, 2010