An

Industry Oriented Mini Project Report On

HAND GESTURE CANVAS - A NEW DIGITAL DRAWING EXPERIENCE

Submitted in partial fulfilment of the requirements for the award of degree

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING
(DATA SCIENCE)

Submitted By

S.SINDHUJA	**217Z1A6751**
A.SRAVANI	**217Z1A6705**
L.SANDEEP	**217Z1A6736**

Under the Guidance of

Mrs. K. Ashwini

Assistant Professor

SCHOOL OF ENGINEERING
Department of Computer Science and Engineering

NALLA NARASIMHA REDDY
EDUCATION SOCIETY'S GROUP OF INSTITUTION
(AN AUTONOMOUS INSTITUTION)

Approved by AICTE, New Delhi, Chowdariguda (V) Korremula 'x' Roads, Via Narapally, Ghatkesar (Mandal)Medchal (Dist), Telangana-500088

2024-2025

NALLA NARASIMHA REDDY
Education Society's Group of Institutions - Integrated Campus
(UGC AUTONOMOUS INSTITUTION)

SCHOOL OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project report titled **"HAND GESTURE CANVAS - A NEW DIGITAL DRAWING EXPERIENCE"** is being submitted by **S.Sindhuja (217Z1A6751) , A.Sravani (217Z1A6705) and L.Sandeep (217Z1A6736)** in partial fulfilment for the award of **Bachelor of technology in Computer Science & Engineering (Data Science)** 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 (Mrs.K.Ashwini)

Head of department (Dr.K.Rameshwaraiah)

Submitted for Viva voce Examination held on ……

External Examiner

DECLARATION

We S.Sindhuja , A.Sravani and L.Sandeep are students of **Bachelor of Technology in Computer Science and Engineering (Data Science) , Nalla Narasimha Reddy Education Society's Group Of Institutions**, Hyderabad, Telangana, hereby declare that the work presented in this project work entitled **Hand Gesture Canvas – A new digital drawing experience** is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of engineering ethics. It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning.

By

S.Sindhuja **217Z1A6751**

A.Sravani **217Z1A6705**

L.Sandeep **217Z1A6736**

Date:

Signature:

ACKNOWLEDGEMENT

We express our sincere gratitude to our guide **Mrs.K.Ashwini,** Assistant Professor in Computer Science and Engineering Department, NNREGSI, who motivated throughout the period of the project and also for her valuable and intellectual suggestions apart from her adequate guidance, constant encouragement right throughout our work.

We would like to express our profound gratitude to our mini project In-charge **Mrs. B. Sriveni,** Assistant Professor Computer Science and Engineering Department, NNREGSI, for her support and guidance in completing our project and for giving us this opportunity to present the project work.

We profoundly express thanks to **Dr. K. Rameshwaraiah,** Head of Computer Science and Engineering Department, NNREGSI, for his cooperation and encouragement in completing the project successfully.

We wish to express our sincere thanks to **Dr. G. Janardhana Raju,** Dean School of Engineering for providing the facilities for completion of the project.

We wish to express our sincere thanks to **Dr**. **C. V. Krishna Reddy,** Director NNRESGI for providing the facilities for completion of the project.

Finally, we would like to thank overall mini-project coordinator, **members of Project Review Committee (PRC),** all the faculty members and supporting staff of the Department of Computer Science and Engineering for extending their help in all circumstances.

By

S.Sindhuja **217Z1A6751**

A.Sravani **217Z1A6705**

L.Sandeep **217Z1A6736**

ABSTRACT

With increasing technology each sector needs to be modernized. With the improvement of clever gadgets, the system can be now controlled virtually with aid of using human gestures. While using paint, sometimes we feel difficult to draw and feel like drawing our imagination just by waving our hand. The Project Hand Gesture Canvas makes a speciality of growing a motion-to-textual converter. This project works on hand tracking system development which aims to track the hand which acts as pen and functioning as pen to create or draw different shapes, write any text and clear using Open Computer Vision Library (OpenCV) and Media-pipe. The development of this project involves the integration of computer vision, machine learning algorithms and augumented reality to detect and interpret hand movements captured by a camera. The system uses a combination of real-time hand tracking and gesture recognition to translate physical gestures into digital drawings. 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. The findings suggest that hand gesture Canvas is intuitive and accessible, with promising applications in education, remote collaboration, and interactive installations. Future improvements aim to enhance accuracy and expand features, potentially transforming digital content interaction.

Keywords: Hand Gesture Recognition, Real-Time Hand Tracking, OpenCV, MediaPipe, Virtual Control.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.INTRODUCTION

1.1 Background

In recent years, advancements in human-computer interaction (HCI) have led to the development of more intuitive and natural ways for users to interact with computers and digital systems. One such development is gesture-based interaction, which allows users to control devices or software applications through hand gestures instead of traditional input devices like a mouse or keyboard. Gesture recognition has become particularly popular in areas such as virtual reality, gaming, sign language interpretation, and creative applications like digital drawing.

In creative industries, artists and designers often rely on physical tools such as styluses, drawing tablets, and touchscreens. However, the integration of gesture recognition with computer vision offers a novel approach to digital drawing that eliminates the need for physical input devices. By leveraging hand-tracking technology, users can draw on a virtual canvas in mid-air using simple hand movements, making the process more accessible, intuitive, and innovative.

This project explores the creation of a gesture-based virtual drawing application using computer vision technologies like OpenCV and Mediapipe. The application allows users to draw on a virtual canvas by moving their hand in front of a webcam, making it suitable for creative and educational purposes.

1.2 Context of the project

The primary motivation behind this project is to enhance the way users interact with digital drawing systems by providing a touch-free, gesture-based interface. Traditional drawing methods, such as using a stylus or mouse, have limitations in terms of freedom of movement, user experience, and accessibility. For example, in situations where users do not have access to specialized drawing equipment, a gesture-based solution using a simple webcam can be a highly accessible alternative.

The project leverages real-time hand tracking to detect the user's hand movements and translate them into drawing commands on a virtual canvas. By integrating hand gestures for color selection, drawing, and clearing the canvas, the application provides an engaging and interactive platform for users. This system can be particularly useful in fields like digital art, design, online education, and virtual collaboration, where the need for intuitive and responsive interaction is growing.

1.3 Objectives

The primary objective of this project is to develop a gesture-based virtual drawing application using hand-tracking technology. The system will allow users to:

- Draw on a virtual canvas by simply moving their hand in front of a camera.
- Select different colors for drawing using simple hand gestures.
- Clear the canvas or start a new drawing with a hand gesture.
- Save the drawing with a keypress.

The goal is to create an interactive and user-friendly application that can track hand movements in real-time and provide a smooth drawing experience.

1.4 Scope

The scope of this project is to develop a functional prototype of a gesture-based virtual drawing system that can be extended and enhanced in the future. The project focuses on the following aspects:

- ✓ Hand Tracking: The system uses Mediapipe to track hand landmarks in real-time, focusing on detecting the user's forefinger and thumb for drawing and control actions.
- ✓ Drawing on Virtual Canvas: Users can draw on the virtual canvas by moving their hand in front of the camera. The system will track the position of the forefinger and translate it into drawing strokes on the canvas.
- ✓ Color Selection: The system provides multiple color options (blue, green, red, yellow) for the user to choose from, allowing for a more dynamic drawing experience.
- ✓ Clear Canvas: Users can clear the entire canvas with a gesture, making it easy to start over without needing to interact with any physical buttons.
- ✓ Real-Time Interaction: The system processes video frames in real-time, ensuring smooth interaction between hand gestures and the drawing application

2. LITERATURE SURVEY

2.1 Introduction

Gesture-based systems have been an area of active research in recent years, particularly in the domain of human-computer interaction. Various systems have been proposed that enable users to control applications using hand gestures. With advancements in computer vision and machine learning, gesture recognition technologies have evolved, allowing for more precise and responsive systems.

2.2 Existing Systems

Several systems currently exist for gesture-based control and drawing applications:

- Leap Motion: A hand-tracking device that detects the motion of fingers and hands in 3D space. However, it requires specialized hardware and may not be as accessible for users without the device.
- Touch-based Drawing Applications: Applications that allow users to draw using touchscreens or styluses, such as Adobe Illustrator or Microsoft Paint. These applications still require physical interaction with the screen, limiting the user's freedom.
- Voice and Motion-Based Systems: Other systems allow users to control devices via voice or motion sensors, but they tend to lack precision when it comes to fine movements, such as drawing.

2.3 Proposed System

In contrast to the existing systems, this project aims to use a webcam combined with computer vision techniques to track hand gestures and draw on a virtual canvas. By using readily available hardware (a standard webcam) and open-source libraries such as OpenCV, Mediapipe, and NumPy, the system eliminates the need for specialized hardware like Leap Motion. The proposed system focuses on the following features:

- Hand Gesture-Based Drawing: Users can draw on a virtual canvas by moving their hand in front of the webcam.
- Real-Time Interaction: The system processes video frames in real-time, ensuring smooth interaction.

- Color Selection and Canvas Control: The system provides a color selection interface, allowing users to choose different colors for drawing. The user can also clear the canvas with a simple hand gesture.
- The advantage of the proposed system is that it does not require any physical contact, making it suitable for various use cases, such as remote collaboration, virtual whiteboarding, and creative applications.

3. SYSTEM ANALYSIS

3.1 Functional Requirements

1. Hand Tracking and Drawing:
   - The system must capture real-time hand movements using a webcam and track finger landmarks using the MediaPipe library.
   - It should allow drawing on a virtual canvas based on finger gestures, specifically detecting the forefinger for drawing.
2. Color Selection:
   - Users can select different drawing colors (blue, green, red, yellow) by moving their hand over corresponding color buttons displayed on the screen.
3. Clear Canvas:
   - The system provides a "Clear" button to reset the entire canvas and remove any previously drawn lines.
4. Save Canvas:
   - Users should be able to save their artwork by pressing the ‘s’ key, generating an image file (PNG) of the drawing.
5. Quit System:
   - The system should allow users to exit the application by pressing the ‘q’ key.

3.2Non-Functional Requirements

1. Performance:
   - The system must operate in real-time, processing hand movements without lag to ensure a smooth drawing experience.
   - It should efficiently handle webcam input and maintain performance during continuous gesture tracking and drawing.’
2. Usability:
   - The user interface must be simple and intuitive, allowing non-experts to easily interact with the system through gestures. Buttons and feedback should be clear, especially the color selection and drawing area.
3. Scalability:
   - The system must efficiently handle large resolutions for canvas drawing, allowing users to create detailed drawings without performance degradation.

4. Accuracy:
   - ❖ Hand tracking should be accurate to ensure that drawing is precise and responds effectively to user gestures.
5. Resource Usage:
   - ❖ The system should use minimal CPU and memory resources while processing video input and hand gestures, ensuring it can run smoothly on common hardware setups.

3.3 Interface Requirements

3.3.1 User Requirements:

- Basic familiarity with computer interaction through gestures and keyboard inputs.
- Understanding of how to interact with a webcam-based application (e.g., how to position hands).

3.3.2 System Requirements:

a. Software:

- ➢ Operating System: Windows, macOS, or Linux
- ➢ Python 3.x
- ➢ Required libraries: OpenCV, Mediapipe, NumPy

b. Hardware:

- ➢ Webcam with at least 720p resolution
- ➢ Dual-core processor (Intel i3 or equivalent)
- ➢ 4GB RAM or higher

4. SYSTEM DESIGN

4.1 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general- purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object oriented computersoftware. In its current form UML is comprised of two major components: a Meta-modeland a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well asfor business modeling and other non-software systems. The UML represents a collectionof best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express thedesign of software projects.

Goals:

The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modeling Language so that they candevelop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.Be independent of particular programming languages and development process. Provide a formal basis for understanding the modeling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks,patterns and components.

4.1.1 Data Flow Diagram

A data flow diagram (DFD) is a visual representation of how data moves through a system. Using symbols like rectangles, circles, and arrows, DFDs illustrate the flow of information, highlighting inputs, outputs, storage points, and the routes between them. These diagrams can have multiple levels of detail, starting with a high-level overview and progressing to more specific subprocesses. There are 2 levels in the below diagram.

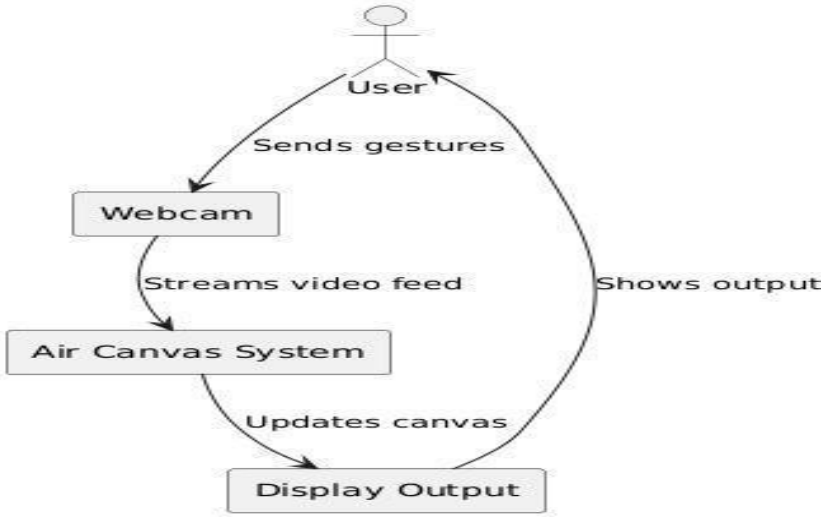

Fig :4.1.1.1 Data Flow Diagram : Level 0

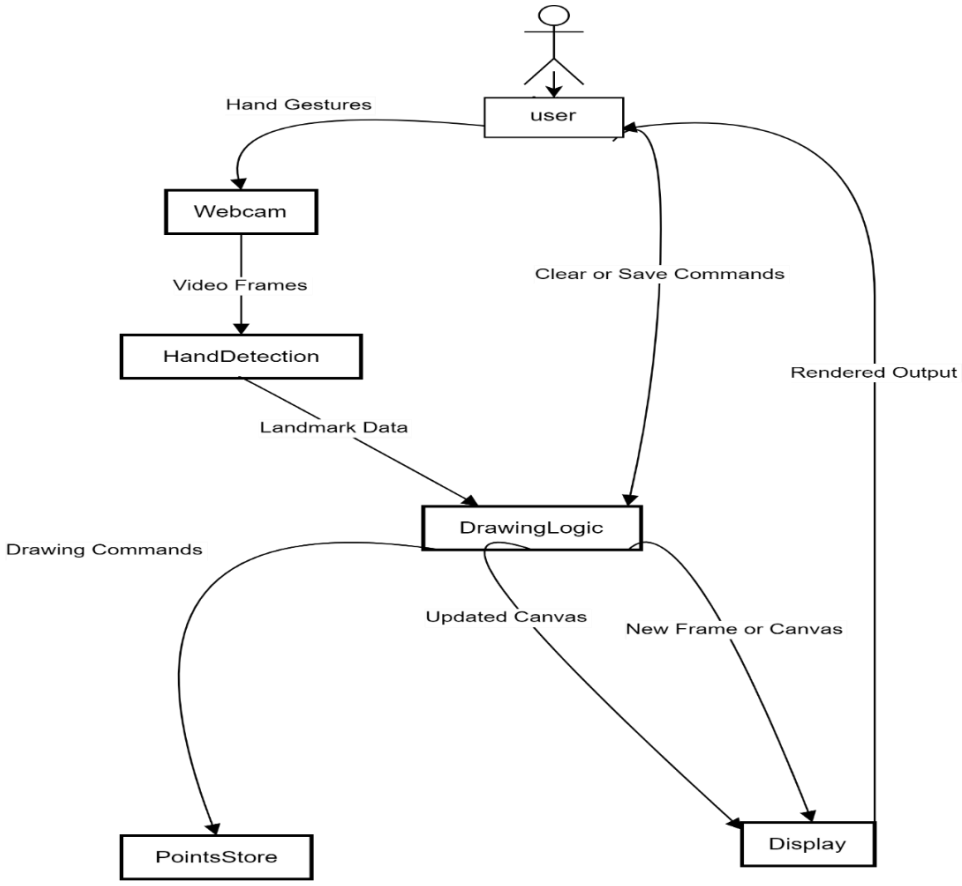

Fig :4.1.1.2 Data Flow Diagram : Level 1

4.1.2 Use Case Diagram:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is topresent a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those usecases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

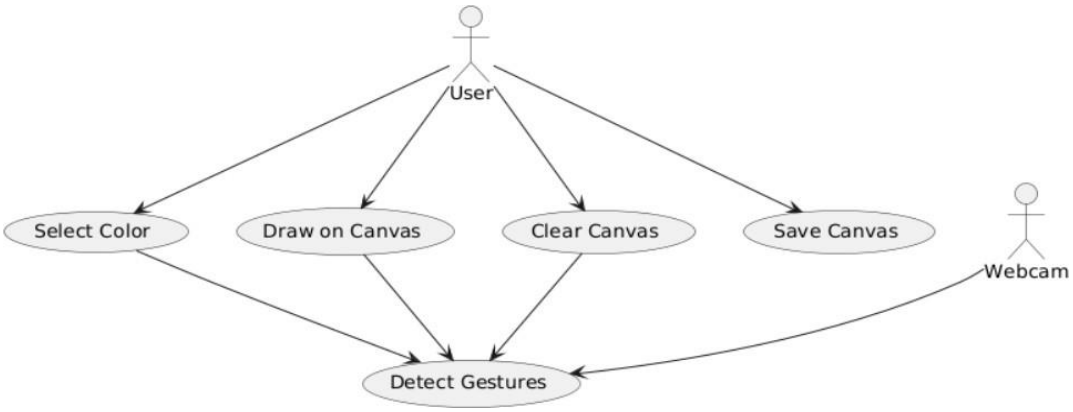

Fig :4.1.2.1 Usecase Diagram

4.1.3 Class Diagram:

In software engineering, a class diagram in the Unified Modeling Language (UML)is a type of static structure diagram that describes the structure of a system by showingthe system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

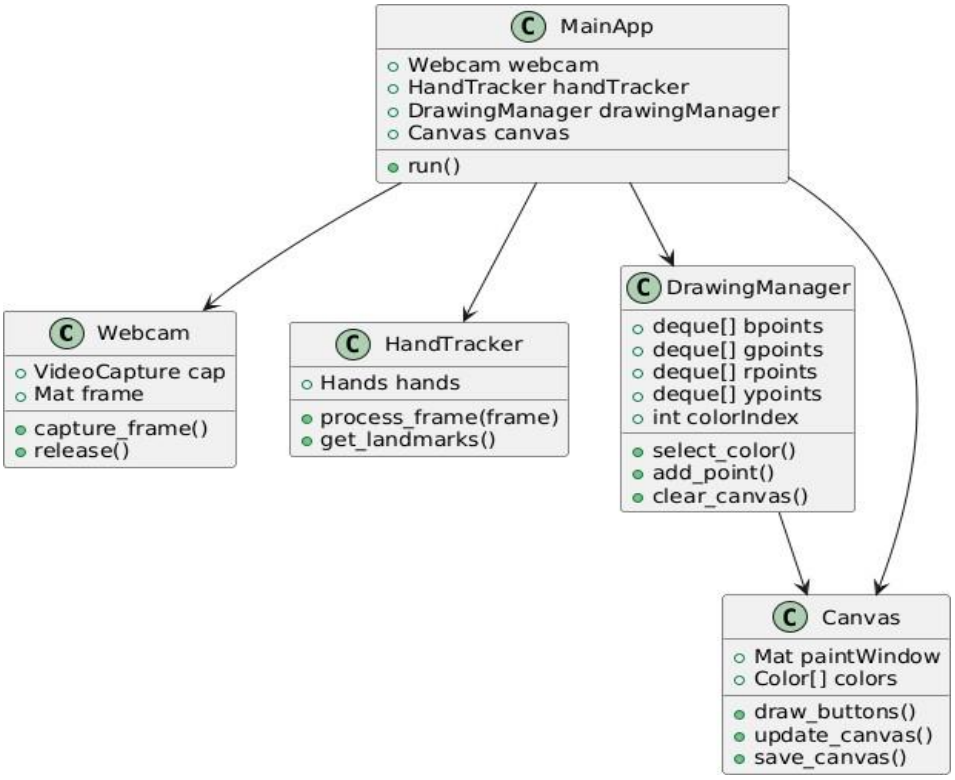

Fig :4.1.3.1 Class Diagram

4.1.4 Sequence Diagram:

A sequence diagram in Unified Modelling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

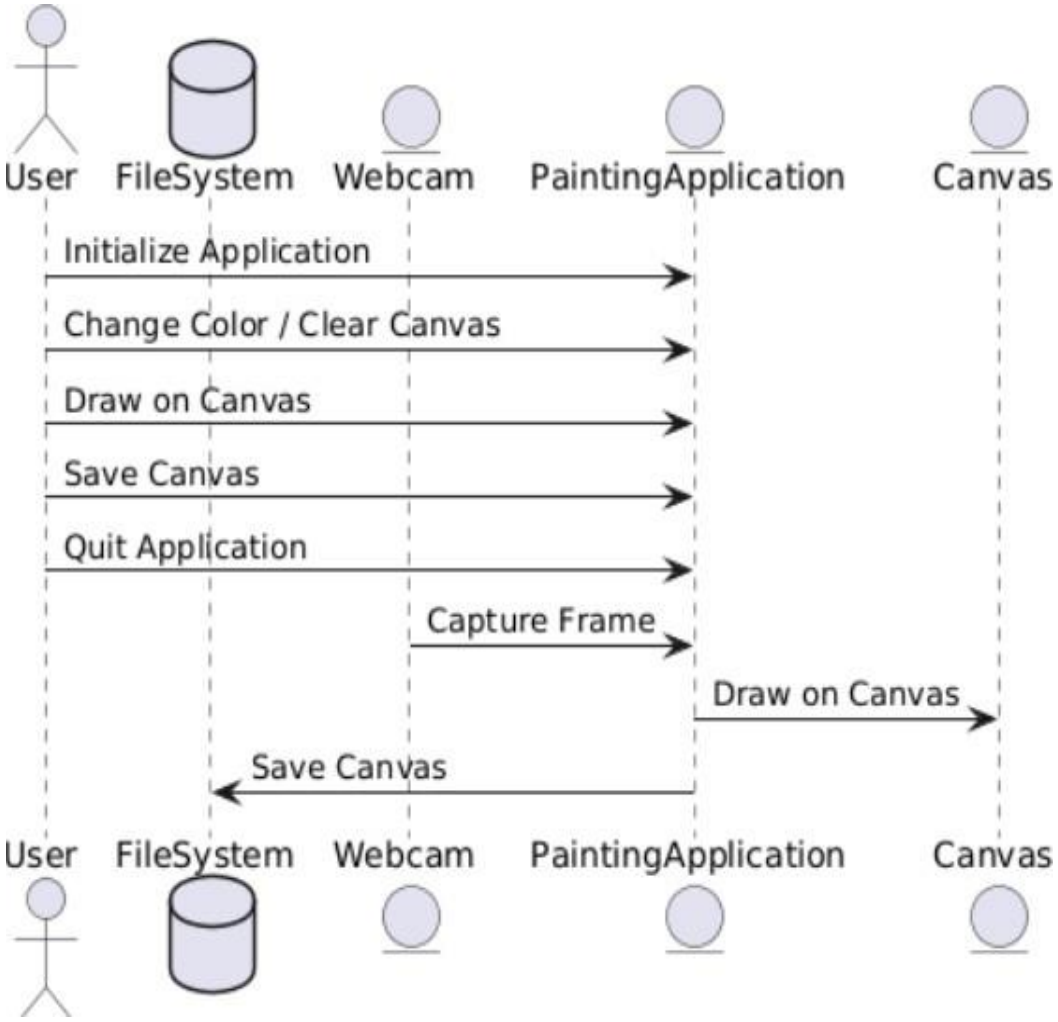

Fig :4.1.4 Sequence Diagram

4.1.5 Activity Diagram:

Activity diagrams are graphical representations of workflows of stepwise activitiesand actions with support for choice, iteration and concurrency. In the Unified ModelingLanguage, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overallflow of control.

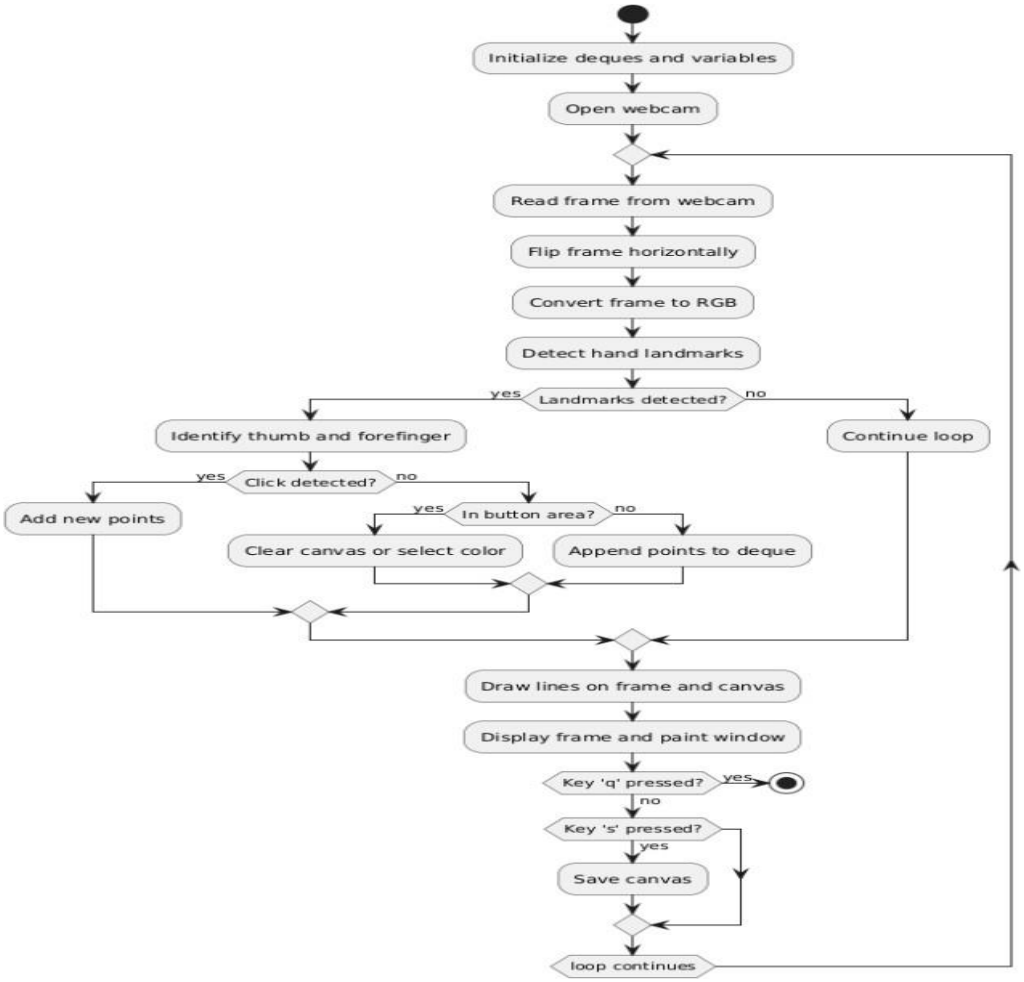

Fig :4.1.5.1 Activity Diagram

5. IMPLEMENTATION & RESULTS

5.1 Method of Implementation

5.1.1 What is Python :-

Python is currently the most widely used multi-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.

Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

The biggest strength of Python is huge collection of standard libraries which can be used for the following –

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox) Image processing (like Opencv, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium) Test frameworks
- Multimedia

Python is a high-level programming language developed by Guido van Rossum in 1991. It emphasizes code readability, has a dynamic type system, automatic memory management, supports multiple programming paradigms, and has a comprehensive standard library. Python is interpretable, allowing direct interaction with the interpreter. It prioritizes speed of development, maintainability, and maintainability. Its large standard library allows for quick implementation, saving time, and easy patching and updating by non-Python background users.

5.1.1 What is Machine Learning : -

Machine learning is a subfield of artificial intelligence that focuses on building models of data. It involves building mathematical models to understand data, which can be adapted to observed data. These models can predict and understand new data aspects. Understanding the problem setting in machine learning is crucial for effective use of these tools. There are various approaches to machine learning, but categorizations can be misleading.

MODULES USED IN PROJECT

1.Open CV

OpenCV (Open Source Computer Vision Library) is a popular library used for real-time computer vision tasks. It provides various tools to capture, process, and manipulate images and videos. When it is integrated with various libraries, such as NumPy, python is capable of processing the openCV array structure for analysis. To Identify an image pattern and its various features we use vector space and perform mathematical operations on these features.

The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it's free for both academic and commercial use.

2.Numpy

Numpy is a general-purpose array-processing package. It provides a high- performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object.
- Sophisticated (broadcasting) functions.
- Tools for integrating C/C++ and Fortran code.
- Useful linear algebra, Fourier transform, and random number capabilities.

Besides its obvious scientific uses, Numpy can also be used as an efficient multi- dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

3.Mediapipe

Mediapipe is a machine learning framework developed by Google that provides various solutions for detecting and tracking objects, including hand landmarks, face detection, and object detection in real-time.

Since 2012, Google has used it internally in several products and services. It was initially developed for real-time analysis of video and audio on YouTube. Gradually it got integrated into many more products; the following are some.

i. Perception system in NestCam

ii. Object detection by Google Lens
iii. Augmented Reality Ads
iv. Google Photos
v. Gmail
vi. Collections

The collections module in Python provides specialized data structures, including deque (double-ended queue), which allows appending and popping elements from both ends with efficient memory management. Efficiently stores drawing points and tracks the user's hand movements for different colors.

It was introduced to improve the functionalities of the built-in collection containers. Python collection module was first introduced in its 2.4 release.

4. Built-In Modules:

- cv2.rectangle():

This function is used to draw rectangles for the color selection buttons (e.g., blue, green, red, yellow) on the virtual canvas. It visually displays the available color options.

- cv2.putText():

This function adds text labels ("CLEAR", "BLUE", "GREEN", etc.) onto the canvas above the color buttons for easy recognition.

The code implements a virtual painting application where users can draw on a digital canvas using hand gestures detected via a webcam. Key elements include:

1. Libraries and Initialization:
    - Uses OpenCV for video handling and drawing, Mediapipe for hand tracking, and Deque to efficiently store drawn points.
    - Initializes variables for tracking points for each color (blue, green, red, yellow) and creates a white canvas with color and clear buttons.
2. Color Buttons:
    - The top section of the canvas includes buttons for color selection (blue, green, red, yellow) and a clear button.
3. Webcam and Hand Detection:
    - The webcam feed is processed using OpenCV, and Mediapipe detects hand

landmarks, focusing on the index finger tip for drawing.

4. Hand Gestures:
    - The index finger is used to draw on the canvas, while proximity between the thumb and index finger signals a new line segment.
    - Hand positions in the button area trigger color selection or canvas clearing.
5. Drawing:
    - Lines are drawn by connecting points based on finger movement, with different colors based on the current selection.
6. Display:
    - The real-time video feed and the drawn canvas are displayed simultaneously.
7. User Interaction:
    - Pressing 'q' exits the application, while 's' saves the current canvas as an image.
8. Cleanup:
    - On exit, the webcam is released, and all windows are closed.

In summary, this application allows real-time drawing on a virtual canvas using simple hand gestures, enabling color selection, line drawing, and canvas management without physical tools.

5.2 EXPLANATION OF KEY FUNCTIONS

Webcam Initialization:

- cv2.VideoCapture(0) starts the webcam for capturing video.
- cv2.flip(frame, 1) flips the video horizontally for intuitive interaction.

Color Space Conversion:

- cv2.cvtColor(frame, cv2.COLOR_BGR2RGB) converts the frame from BGR to RGB, which is required by Mediapipe for hand tracking.

Hand Tracking:

- mp.solutions.hands.Hands() initializes hand tracking.
- hands.process(framergb) processes the frame to detect hand landmarks.

Drawing Hand Landmarks:

- mpDraw.draw_landmarks() draws landmarks (e.g., fingertips) and connections

between them on the frame.

- cv2.circle() highlights the fingertip for user feedback.

Storing Points for Drawing:

- deque() stores drawing points for each color, enabling continuous drawing.

Buttons and Interaction:

- cv2.rectangle() draws interactive buttons (Clear, Blue, Green, Red, Yellow) on the screen.
- cv2.putText() labels the buttons for clarity.

Drawing Lines:

- cv2.line() draws lines between consecutive points, simulating hand-drawn strokes.

Displaying and Interaction:

- cv2.imshow() shows the live webcam feed and the virtual canvas.
- cv2.waitKey(1) checks for key presses like 'q' to quit or 's' to save the drawing.

Saving and Cleanup:

- cv2.imwrite() saves the drawn canvas as an image when the user presses 's'.
- cap.release() and cv2.destroyAllWindows() release resources and close the application.

5.3 SOURCE CODE

```
import cv2 # Importing OpenCV library for computer vision tasks
import numpy as np # Importing NumPy for numerical operations
import mediapipe as mp # Importing Mediapipe for hand tracking
from collections import deque # Importing deque from collections to handle drawing points
# Initialize deques to store points for different colors
bpoints = [deque(maxlen=1024)] # Blue points
gpoints = [deque(maxlen=1024)] # Green points
rpoints = [deque(maxlen=1024)] # Red points
ypoints = [deque(maxlen=1024)] # Yellow points
# Indexes to keep track of points in different color arrays
blue_index = 0
green_index = 0
```

```
red_index = 0
yellow_index = 0
# Kernel for image dilation
kernel = np.ones((5, 5), np.uint8)
# List of colors in BGR format
colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (0, 255, 255)]
colorIndex = 0 # Default color index
# Set up the canvas window with color buttons
paintWindow = np.zeros((471, 636, 3)) + 255 # White canvas
paintWindow = cv2.rectangle(paintWindow, (40, 1), (140, 65), (0, 0, 0), 2) # Clear button
paintWindow = cv2.rectangle(paintWindow, (160, 1), (255, 65), (255, 0, 0), 2) # Blue button
paintWindow = cv2.rectangle(paintWindow, (275, 1), (370, 65), (0, 255, 0), 2) # Green button
paintWindow = cv2.rectangle(paintWindow, (390, 1), (485, 65), (0, 0, 255), 2) # Red button
paintWindow = cv2.rectangle(paintWindow, (505, 1), (600, 65), (0, 255, 255), 2) # Yellow button
# Adding text labels on the canvas buttons
cv2.putText(paintWindow, "CLEAR", (49, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "BLUE", (185, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "GREEN", (298, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "RED", (420, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "YELLOW", (520, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
# Creating a window named 'Paint'
cv2.namedWindow('Paint', cv2.WINDOW_AUTOSIZE)
# Initialize Mediapipe for hand tracking
mpHands = mp.solutions.hands
```

```
hands = mpHands.Hands(max_num_hands=1, min_detection_confidence=0.7)
mpDraw = mp.solutions.drawing_utils # Utility for drawing hand landmarks
#  Initialize the webcam
cap = cv2.VideoCapture(0)
ret = True
save_counter = 0 # Counter to keep track of saved images
while ret:
    # Read each frame from the webcam
    ret, frame = cap.read()
    x, y, c = frame.shape
    # Flip the frame horizontally for natural (mirror-like) interaction
    frame = cv2.flip(frame, 1)
    # Convert the frame from BGR to RGB
    framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
 # Draw color buttons on the frame
    frame = cv2.rectangle(frame, (40, 1), (140, 65), (0, 0, 0), 2)
    frame = cv2.rectangle(frame, (160, 1), (255, 65), (255, 0, 0), 2)
    frame = cv2.rectangle(frame, (275, 1), (370, 65), (0, 255, 0), 2)
    frame = cv2.rectangle(frame, (390, 1), (485, 65), (0, 0, 255), 2)
    frame = cv2.rectangle(frame, (505, 1), (600, 65), (0, 255, 255), 2)
    cv2.putText(frame, "CLEAR", (49, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
0, 0), 2, cv2.LINE_AA)
    cv2.putText(frame, "BLUE", (185, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
0, 0), 2, cv2.LINE_AA)
    cv2.putText(frame, "GREEN", (298, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(0, 0, 0), 2, cv2.LINE_AA)
    cv2.putText(frame, "RED", (420, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
0, 0), 2, cv2.LINE_AA)
    cv2.putText(frame, "YELLOW", (520, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(0, 0, 0), 2, cv2.LINE_AA)
# Get hand landmark predictions
    result = hands.process(framergb)
 # Post-process the result
    if result.multi_hand_landmarks:
```

```
        landmarks = []
        for handslms in result.multi_hand_landmarks:
            for lm in handslms.landmark:
                # Convert normalized coordinates to pixel coordinates
                lmx = int(lm.x * 640)
                lmy = int(lm.y * 480)
                landmarks.append([lmx, lmy])
            # Draw hand landmarks on the frame
            mpDraw.draw_landmarks(frame, handslms, mpHands.HAND_CONNEC-
TIONS)
        # Get coordinates of the forefinger and thumb
        fore_finger = (landmarks[8][0], landmarks[8][1])
        center = fore_finger
        thumb = (landmarks[4][0], landmarks[4][1])
        cv2.circle(frame, center, 3, (0, 255, 0), -1) # Draw a circle at the tip of the forefin-
ger
        # Check if thumb and forefinger are close to each other (indicating a click)
        if (thumb[1] - center[1] < 30):
            # Append new deque for each color to start a new line segment
            bpoints.append(deque(maxlen=512))
            blue_index += 1
            gpoints.append(deque(maxlen=512))
            green_index += 1
            rpoints.append(deque(maxlen=512))
            red_index += 1
            ypoints.append(deque(maxlen=512))
            yellow_index += 1
        # Check if the forefinger is within the top button area (for clearing or color selec-
tion)
        elif center[1] <= 65:
            if 40 <= center[0] <= 140: # Clear Button
                # Reset all points and indexes
                bpoints = [deque(maxlen=512)]
                gpoints = [deque(maxlen=512)]
```

```
            rpoints = [deque(maxlen=512)]
            ypoints = [deque(maxlen=512)]
            blue_index = 0
            green_index = 0
            red_index = 0
            yellow_index = 0
            # Clear the paint window
            paintWindow[67:, :, :] = 255
        elif 160 <= center[0] <= 255:
            colorIndex = 0 # Blue
        elif 275 <= center[0] <= 370:
            colorIndex = 1 # Green
        elif 390 <= center[0] <= 485:
            colorIndex = 2 # Red
        elif 505 <= center[0] <= 600:
            colorIndex = 3 # Yellow
    else:
        # Append points to the corresponding deque based on selected color
        if colorIndex == 0:
            bpoints[blue_index].appendleft(center)
        elif colorIndex == 1:
            gpoints[green_index].appendleft(center)
        elif colorIndex == 2:
            rpoints[red_index].appendleft(center)
        elif colorIndex == 3:
            ypoints[yellow_index].appendleft(center)
else:
    # Append new deque for each color to avoid messing up when no hand is detected
    bpoints.append(deque(maxlen=512))
    blue_index += 1
    gpoints.append(deque(maxlen=512))
    green_index += 1
    rpoints.append(deque(maxlen=512))
    red_index += 1
```

```
            ypoints.append(deque(maxlen=512))
            yellow_index += 1
        # Draw lines of all the colors on the canvas and frame
        points = [bpoints, gpoints, rpoints, ypoints]
        for i in range(len(points)):
            for j in range(len(points[i])):
                for k in range(1, len(points[i][j])):
                    if points[i][j][k - 1] is None or points[i][j][k] is None:
                        continue
                    # Draw lines on the frame
                    cv2.line(frame, points[i][j][k - 1], points[i][j][k], colors[i], 2)
                    # Draw lines on the paint window
                    cv2.line(paintWindow, points[i][j][k - 1], points[i][j][k], colors[i], 2)
        # Display the frame and paint window
        cv2.imshow("Output", frame)
        cv2.imshow("Paint", paintWindow)
        # Break the loop if 'q' key is pressed
        key = cv2.waitKey(1)
        if key == ord('q'):
            break
        # Save the canvas if 's' key is pressed
        elif key == ord('s'):
            save_counter += 1
            filename = f"AirCanvas_{save_counter}.png"
            cv2.imwrite(filename, paintWindow)
            print(f"Canvas saved as {filename}")
    # Release the webcam and destroy all active windows
    cap.release()
    cv2.destroyAllWindows()
```

5.4RESULTS

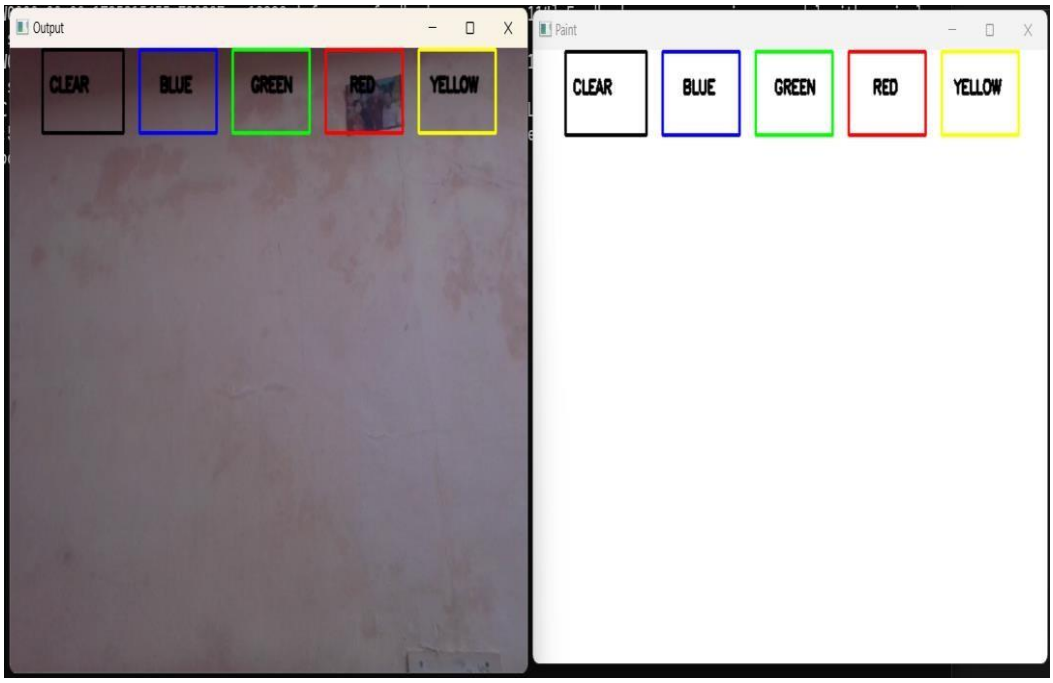

Fig : 5.4.1 Initial display

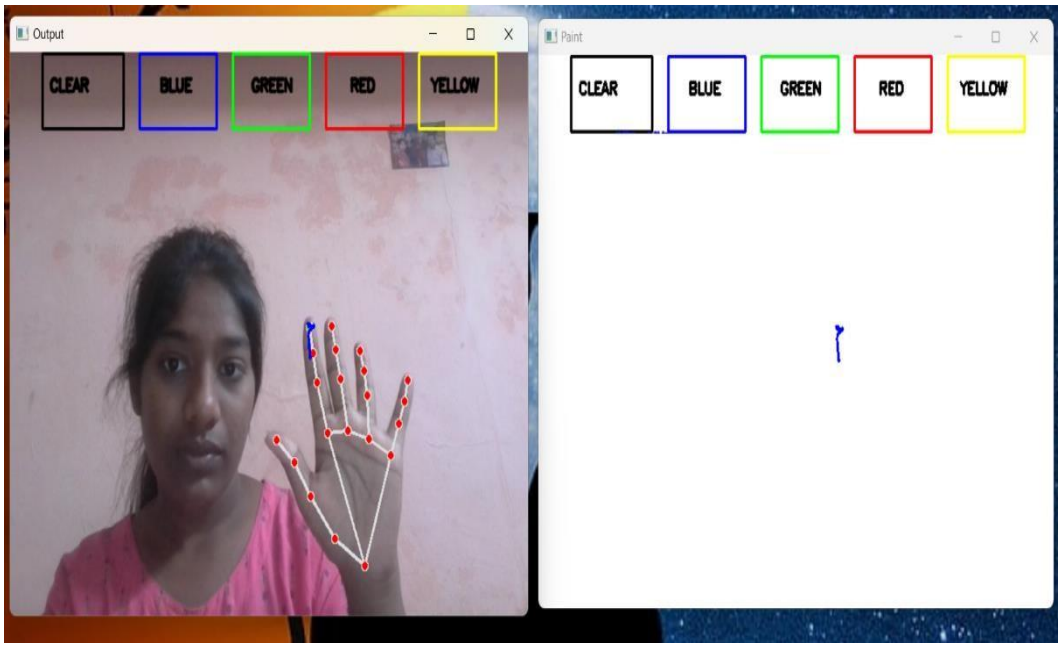

Fig : 5.4.2 Hand tracking

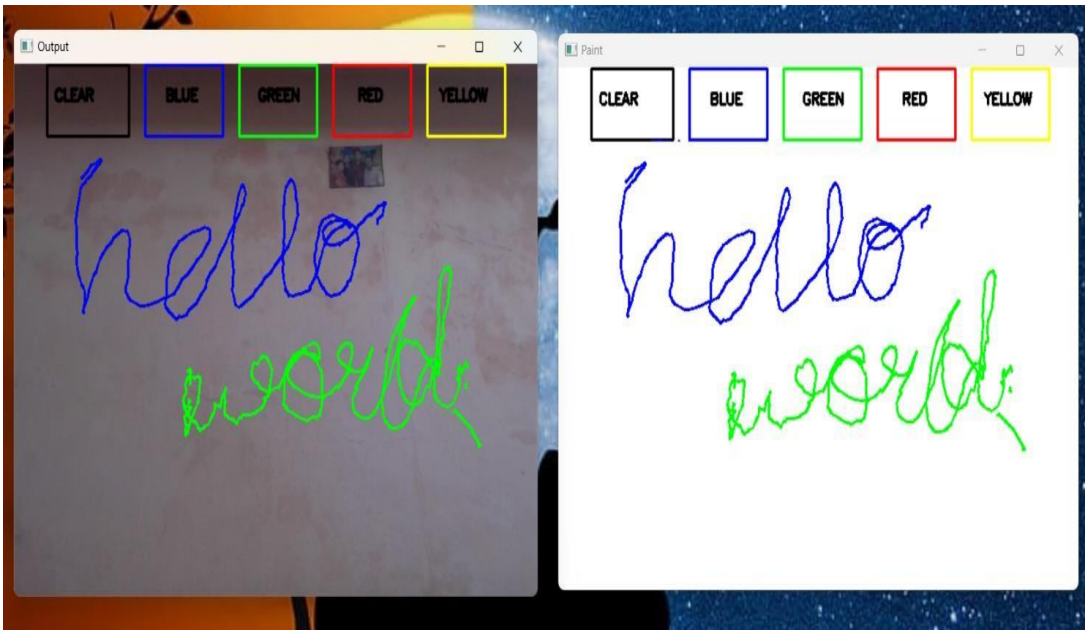

Fig : 5.4.3 Writing on Screen

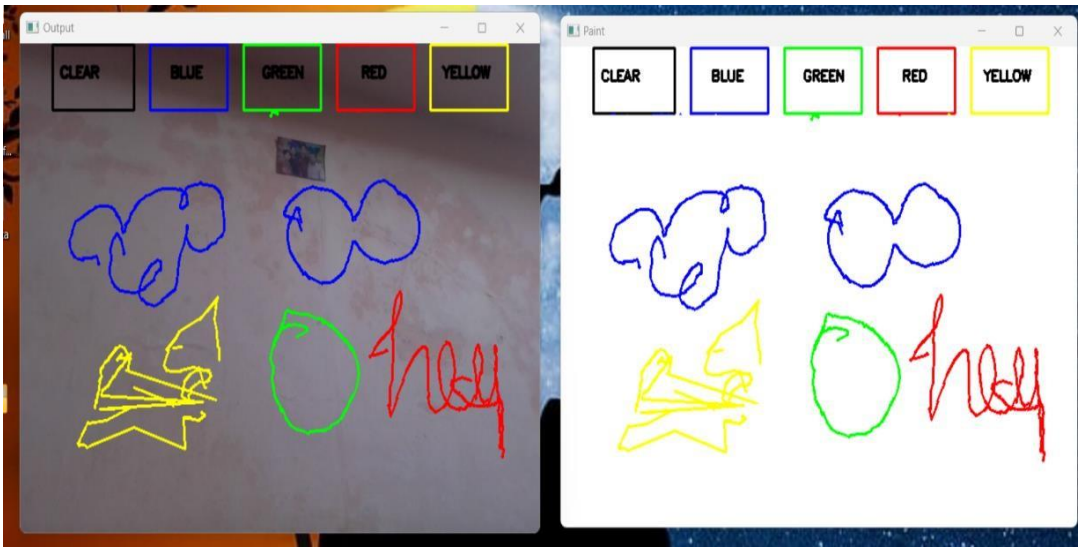

Fig : 5.4.4 Switching between different colors

6. SYSTEM 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.

6.1 TYPES OF TESTS

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.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input **:** Identified classes of valid input must be accepted.

Invalid Input **:** Identified classes of invalid input must be rejected.

Functions **:** Identified functions must be exercised.

Output **:** Identified classes of application outputs must be exercised

Systems/Procedures **:** Interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

This testing the software without any knowledge of the inner workings, structure or language of the module being tested. These tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

6.2 VARIOUS TESTCASE SCENARIOS

Fig : 6.2.1 Testcases

Test Case ID	Description	Input	Expected Output	Actual Output	Status
TC01	Verify application initializes correctly	Start the application	Application window opens with color buttons and canvas	Application window opens with color buttons and canvas	Success
TC02	Verify clear button functionality	Click the clear button on the canvas	Paint window should be cleared, reset drawing area	Paint window cleared, reset drawing area	Success
TC03	Verify drawing with Blue color	Select Blue color, draw on the canvas	Lines drawn on the canvas in Blue color	Lines drawn on the canvas in Blue color	Success
TC04	Verify drawing with Green color	Select Green color, draw on the canvas	Lines drawn on the canvas in Green color	Lines drawn on the canvas in Green color	Success
TC05	Verify drawing with Red color	Select Red color, draw on the canvas	Lines drawn on the canvas in Red color	Lines drawn on the canvas in Red color	Success
TC06	Verify drawing with Yellow color	Select Yellow color, draw on the canvas	Lines drawn on the canvas in Yellow	Lines drawn on the canvas in Yellow	Success

TC07	Verify saving the drawing	Press the 's' key after drawing	Drawing should be saved as an image	Drawing saved as an image	Success
TC08	Verify application closes correctly	Press the 'q' key to close the application	Application should close	Application closed successfully	Success
TC09	Verify hand tracking starts	Start application with a webcam	Hand landmarks should be detected and displayed on the frame	Hand landmarks detected and displayed on the frame	Success
TC10	Verify selecting color via hand movement	Move index finger to hover over color buttons	Selected color should change based on hovering	Selected color changed successfully	Success

7.CONCLUSION AND FUTURE ENHANCEMENT

7.1 Project Conclusion

In this project, we successfully developed a gesture-based virtual drawing system using real-time hand-tracking and augmented reality (AR) techniques. The system allows users to interact with a virtual canvas through natural hand gestures, without the need for physical input devices like a mouse or stylus. By utilizing a standard webcam, OpenCV for video processing, Mediapipe for hand tracking, and NumPy for canvas creation, the system demonstrates how accessible tools can create an intuitive and interactive experience.

The system's ability to recognize hand gestures and map them to drawing actions provides a seamless user experience. Users can select different colors, clear the canvas, and create drawings simply by moving their hands in the air. Furthermore, the integration of AR features enhances the user experience by blending virtual drawings with the real-world environment, making it suitable for creative applications such as digital art, virtual whiteboarding, and remote collaboration.

7.2 Future Enhancement

While the current system is functional and effective, there are several areas for potential improvement and future expansion.

- Advanced Gesture Recognition: The system could be expanded to support more complex gestures, allowing users to perform actions such as undoing, redoing, or resizing drawings.
- Multi-Hand Tracking: Implementing support for multi-hand tracking would enable more intricate interactions, such as two-handed drawing or manipulating objects in the virtual space.
- Cloud Integration: Adding cloud storage features would allow users to save and share their drawings seamlessly across devices.
- Enhanced Augmented Reality: In the future, the system could incorporate more advanced AR features, such as 3D object drawing or integration with virtual reality platforms for a fully immersive experience.

8. REFERENCES

8.1 WEBSITES

- www.google.com
- www.geeksforgeeks.com
- En.wikipedia.org
- Ieeexplore.ieee.org
- www.youtube.com
- www.techvidvan.com

8.2 PAPER REFERENCES

[1] Sonal Gupta, Dr. Satish Kumar, "Air-Writing Recognition Using Machine Learning Techniques," International Journal of Advanced Research in Computer Science and Software Engineering, Volume 8, Issue 4, April 2018.

[2] T. Zhang, S. Y. Huang, "Gesture-Based Human-Computer Interaction Using Convolutional Neural Networks," International Conference on Machine Vision and Information Technology (MVIT), October 2019.

[3]. Y. Huang, X. Liu, X. Zhang, and L. Jin, "A Pointing Gesture Based Egocentric Interaction System: Dataset, Approach, and Application," 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Las Vegas, NV, pp. 370-377, 2016.

[4]. P. Ramasamy, G. Prabhu, and R. Srinivasan, "An economical air writing system is converting finger movements to text using a web camera," 2016 International Conference on Recent Trends in Information Technology (ICRTIT), Chennai, pp. 1-6, 2016.

[5]. Saira Beg, M. Fahad Khan and Faisal Baig, "Text Writing in Air," Journal of Information Display Volume 14, Issue 4, 2013