Chapter

**Virtual Paint**

**AIM:**

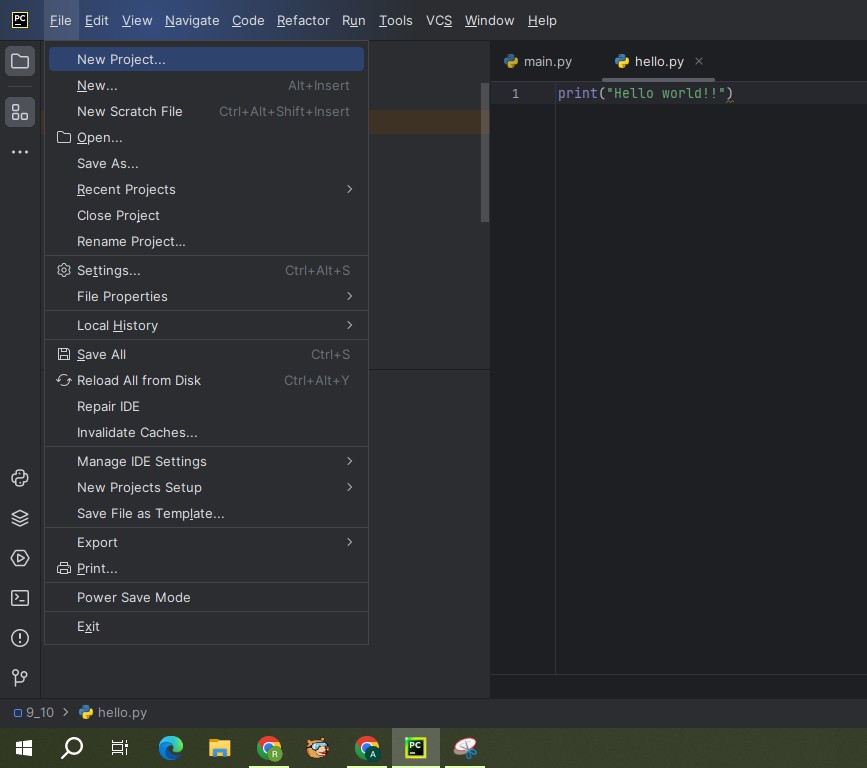
The aim of a "Virtual Paint" project using AI and computer vision technologies like OpenCV is to create an application that allows users to draw on a digital canvas in real-time by simply moving an object, such as a colored marker or their finger, in front of a webcam. The system tracks the movement of the object and translates it into drawing commands, thereby enabling a user to paint virtually without touching a physical medium.

**Introduction:**

The "Virtual Paint" project leverages computer vision and AI to enable users to draw on a digital canvas in real-time using a colored marker or their finger. This innovative application tracks the movement of the object through a webcam and translates it into drawing commands, providing a touchless drawing experience. Key components include object detection, gesture recognition, and real-time drawing capabilities. The project aims to create an intuitive and user-friendly interface for seamless interaction. This technology showcases the potential of AI and computer vision in enhancing digital creativity.

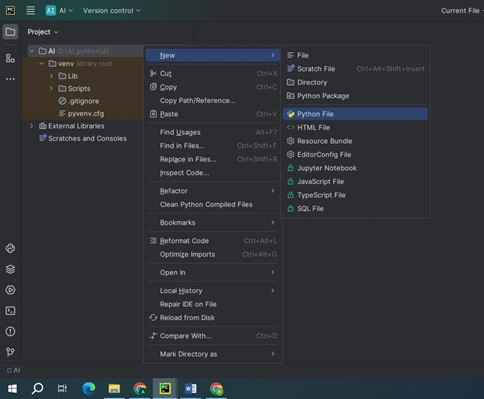
**Procedure:**

* Open PyCharm IDE software.
* Go to Menu —> File —> New Project.

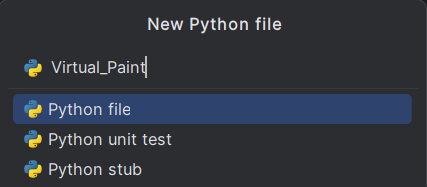


* A screenshot of a program

  Description automatically generatedA window will be appeared as below. Change the Name and Location of the project as per our requirement. Select custom environment for interpreter type. Select Generate new for Environment. Select Virtualenv for Type. Select latest version of python for Base python i.e., Python 3.11.6. And click on create
* Once our project is created, right click on project. Go to —> New —> Python File.



* Give the name to the python file: Virtual\_Paint.



* Once the file is created, copy below given code to Virtual Paint.

**Virtual Mouse** **code:**

import numpy as np

import cv2

from collections import deque

def setValues(x):

print("")

# Creating the trackbars needed for adjusting the marker colour

cv2.namedWindow("Color detectors")

cv2.createTrackbar("Upper Hue", "Color detectors", 153, 180,setValues)

cv2.createTrackbar("Upper Saturation", "Color detectors", 255, 255,setValues)

cv2.createTrackbar("Upper Value", "Color detectors", 255, 255,setValues)

cv2.createTrackbar("Lower Hue", "Color detectors", 64, 180,setValues)

cv2.createTrackbar("Lower Saturation", "Color detectors", 72, 255,setValues)

cv2.createTrackbar("Lower Value", "Color detectors", 49, 255,setValues)

# Giving different arrays to handle colour points of different colours

bpoints = [deque(maxlen=1024)]

gpoints = [deque(maxlen=1024)]

rpoints = [deque(maxlen=1024)]

ypoints = [deque(maxlen=1024)]

#assigning index values

blue\_index = 0

green\_index = 0

red\_index = 0

yellow\_index = 0

kernel = np.ones((5,5),np.uint8)

colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (0, 255, 255)]

colorIndex = 0

#starting the painting window setup

paintWindow = np.zeros((471,636,3)) + 255

paintWindow = cv2.rectangle(paintWindow, (40,1), (140,65), (0,0,0), 2)

paintWindow = cv2.rectangle(paintWindow, (160,1), (255,65), colors[0], -1)

paintWindow = cv2.rectangle(paintWindow, (275,1), (370,65), colors[1], -1)

paintWindow = cv2.rectangle(paintWindow, (390,1), (485,65), colors[2], -1)

paintWindow = cv2.rectangle(paintWindow, (505,1), (600,65), colors[3], -1)

cv2.putText(paintWindow, "CLEAR", (49, 33), cv2.FONT\_HERSHEY\_DUPLEX, 0.5, (0, 0, 0), 2, cv2.LINE\_AA)

cv2.putText(paintWindow, "BLUE", (185, 33), cv2.FONT\_ITALIC, 0.5, (255, 255, 255), 2, cv2.LINE\_AA)

cv2.putText(paintWindow, "GREEN", (298, 33), cv2.FONT\_ITALIC, 0.5, (255, 255, 255), 2, cv2.LINE\_AA)

cv2.putText(paintWindow, "RED", (420, 33), cv2.FONT\_ITALIC, 0.5, (255, 255, 255), 2, cv2.LINE\_AA)

cv2.putText(paintWindow, "YELLOW", (520, 33), cv2.FONT\_ITALIC, 0.5, (150,150,150), 2, cv2.LINE\_AA)

cv2.namedWindow('Paint', cv2.WINDOW\_AUTOSIZE)

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

#Flipping the frame just for convenience

frame = cv2.flip(frame, 1)

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

u\_hue = cv2.getTrackbarPos("Upper Hue", "Color detectors")

u\_saturation = cv2.getTrackbarPos("Upper Saturation", "Color detectors")

u\_value = cv2.getTrackbarPos("Upper Value", "Color detectors")

l\_hue = cv2.getTrackbarPos("Lower Hue", "Color detectors")

l\_saturation = cv2.getTrackbarPos("Lower Saturation", "Color detectors")

l\_value = cv2.getTrackbarPos("Lower Value", "Color detectors")

Upper\_hsv = np.array([u\_hue,u\_saturation,u\_value])

Lower\_hsv = np.array([l\_hue,l\_saturation,l\_value])

frame = cv2.rectangle(frame, (40,1), (140,65), (122,122,122), -1)

frame = cv2.rectangle(frame, (160,1), (255,65), colors[0], -1)

frame = cv2.rectangle(frame, (275,1), (370,65), colors[1], -1)

frame = cv2.rectangle(frame, (390,1), (485,65), colors[2], -1)

frame = cv2.rectangle(frame, (505,1), (600,65), colors[3], -1)

cv2.putText(frame, "CLEAR ALL", (49, 33), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE\_AA)

cv2.putText(frame, "BLUE", (185, 33), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE\_AA)

cv2.putText(frame, "GREEN", (298, 33), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE\_AA)

cv2.putText(frame, "RED", (420, 33), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE\_AA)

cv2.putText(frame, "YELLOW", (520, 33), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (150,150,150), 2, cv2.LINE\_AA)

Mask = cv2.inRange(hsv, Lower\_hsv, Upper\_hsv)

Mask = cv2.erode(Mask, kernel, iterations=1)

Mask = cv2.morphologyEx(Mask, cv2.MORPH\_OPEN, kernel)

Mask = cv2.dilate(Mask, kernel, iterations=1)

cnts,\_ = cv2.findContours(Mask.copy(), cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

center = None

# Ifthe contours are formed

if len(cnts) > 0:

# sorting the contours to find biggest contour

cnt = sorted(cnts, key = cv2.contourArea, reverse = True)[0]

# Get the radius of the enclosing circle around the found contour

((x, y), radius) = cv2.minEnclosingCircle(cnt)

# Draw the circle around the contour

cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)

# Calculating the center of the detected contour

M = cv2.moments(cnt)

center = (int(M['m10'] / M['m00']), int(M['m01'] / M['m00']))

#checking if any button above the screen is clicked/cursor hovered to

if center[1] <= 65:

if 40 <= center[0] <= 140: # Clear Button

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

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 :

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:

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

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

cv2.line(frame, points[i][j][k - 1], points[i][j][k], colors[i], 2)

cv2.line(paintWindow, points[i][j][k - 1], points[i][j][k], colors[i], 2)

cv2.imshow("Tracking", frame)

cv2.imshow("Paint", paintWindow)

cv2.imshow("mask",Mask)

if cv2.waitKey(1) & 0xFF == ord("q"):

break

# Release the camera and all resources

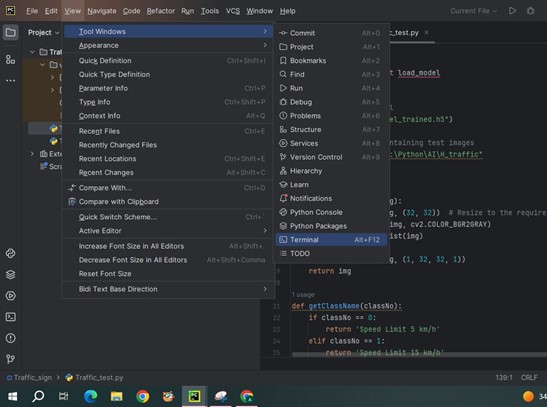
cap.release()

cv2.destroyAllWindows()

**Libraries to install:**

Ensure you have the following libraries installed before running the code:

Go to Menu —> View —> Tool Windows —> Terminal



* **OpenCV**:

OpenCV (cv2) is a Python library for computer vision tasks, offering tools for image and video processing. It supports reading, writing, and manipulating images and videos, along with features like object detection and facial recognition. OpenCV's versatility makes it suitable for a wide range of applications, from basic image operations to advanced machine learning integration. Its cross-platform nature and extensive documentation make it accessible for developers aiming to incorporate vision-based functionalities into their projects efficiently.Install to type the below command in terminal:

pip install opencv



* **NumPy**:

NumPy (Numerical Python) is a fundamental library for scientific computing in Python. It provides support for arrays, matrices, and a large collection of mathematical functions to operate on these data structures efficiently. NumPy is widely used in data analysis, machine learning, and scientific research due to its powerful capabilities and high performance. Install to type the below command in terminal:

pip install NumPy

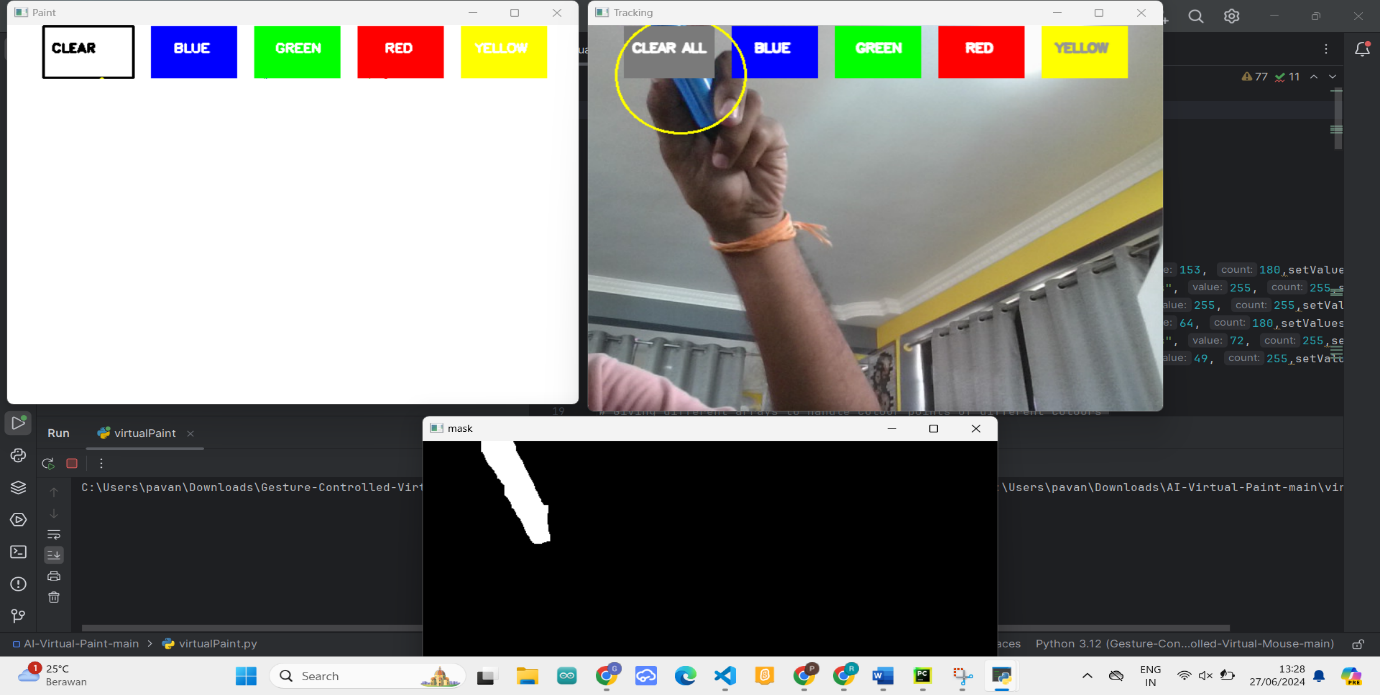
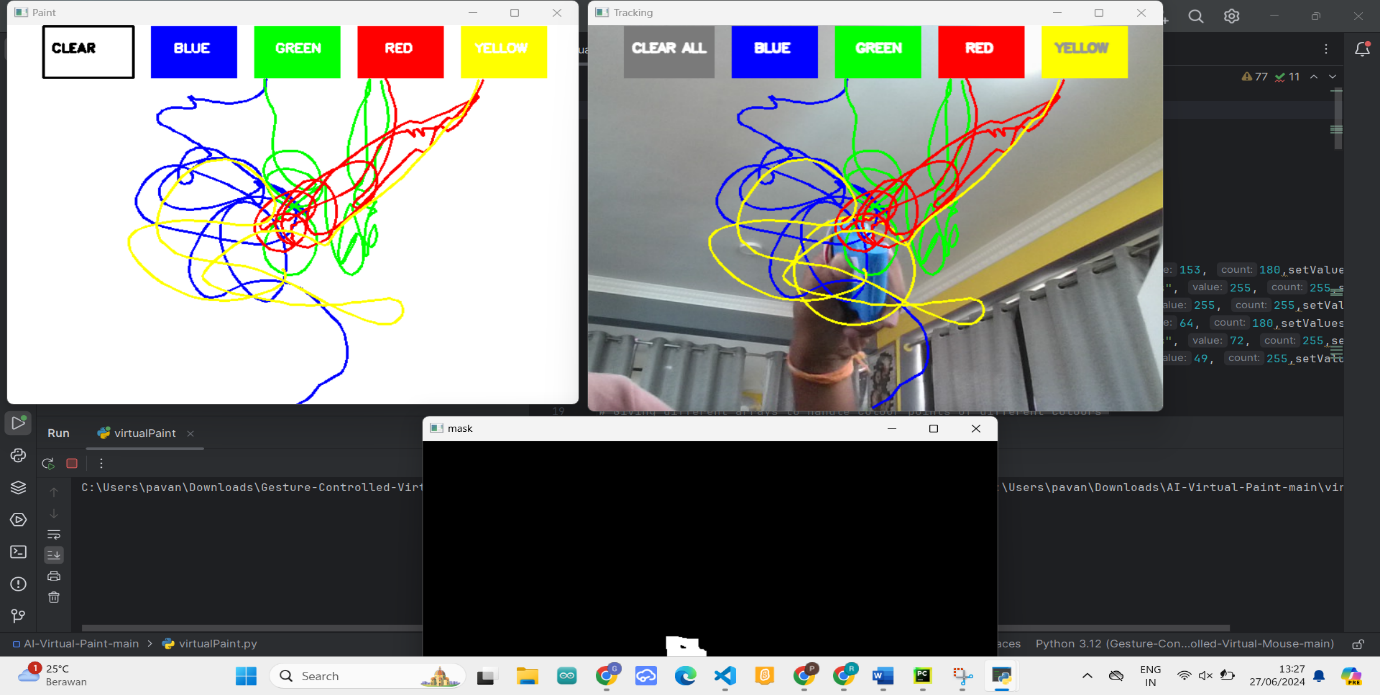
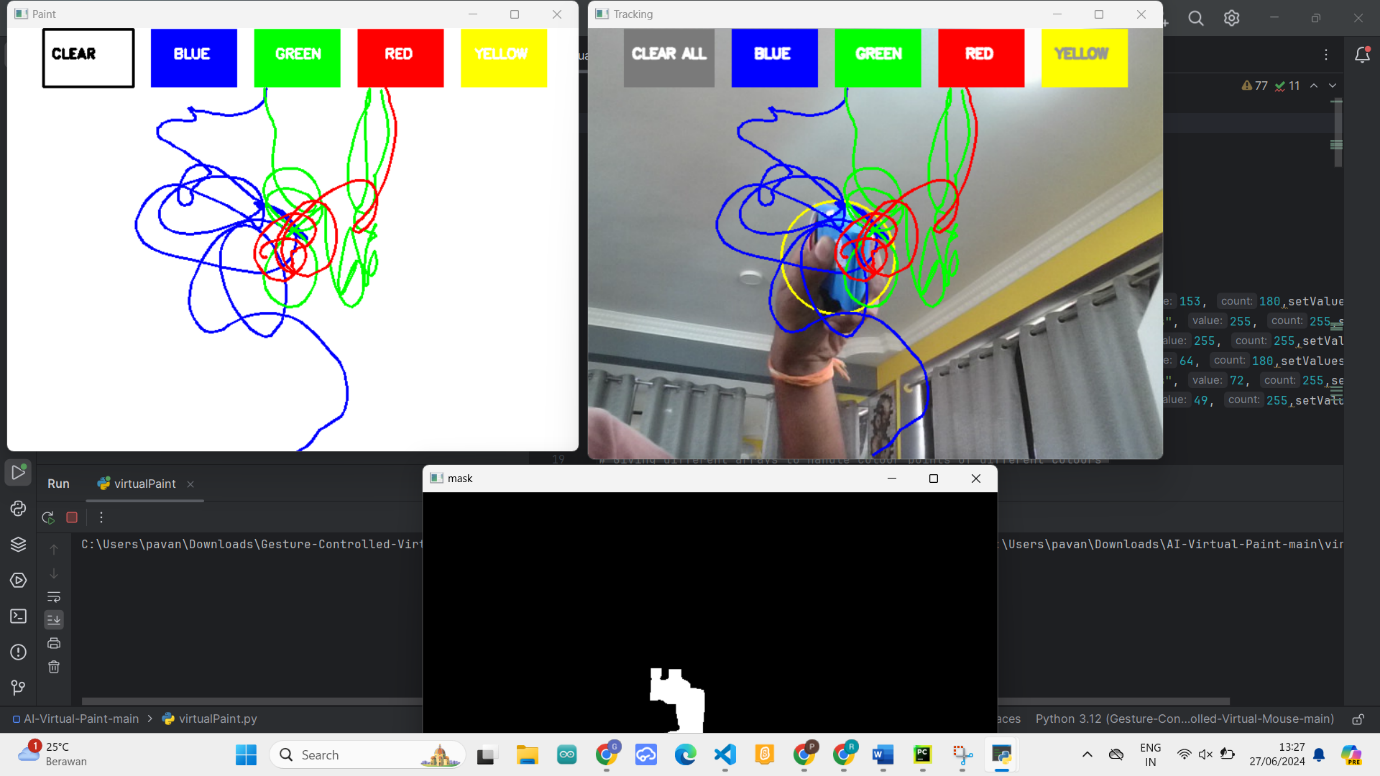
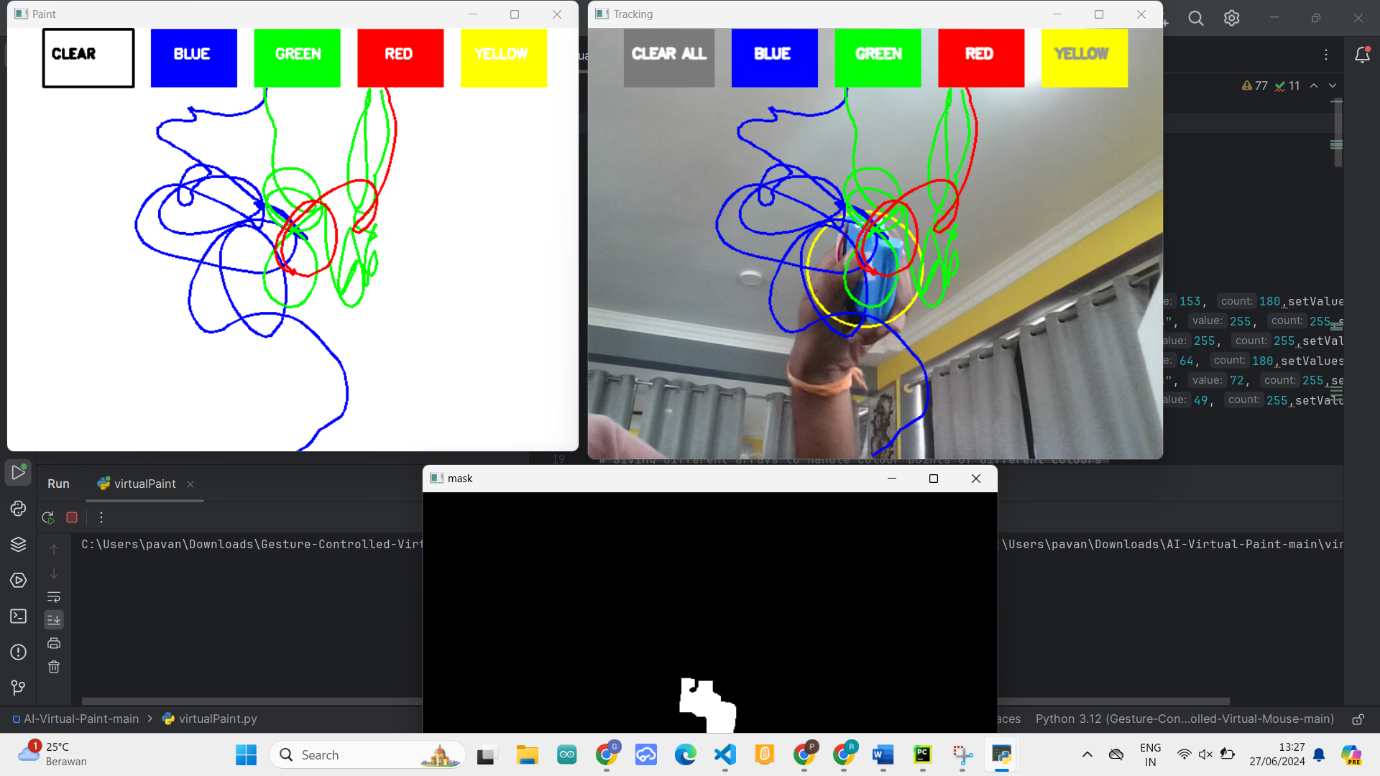
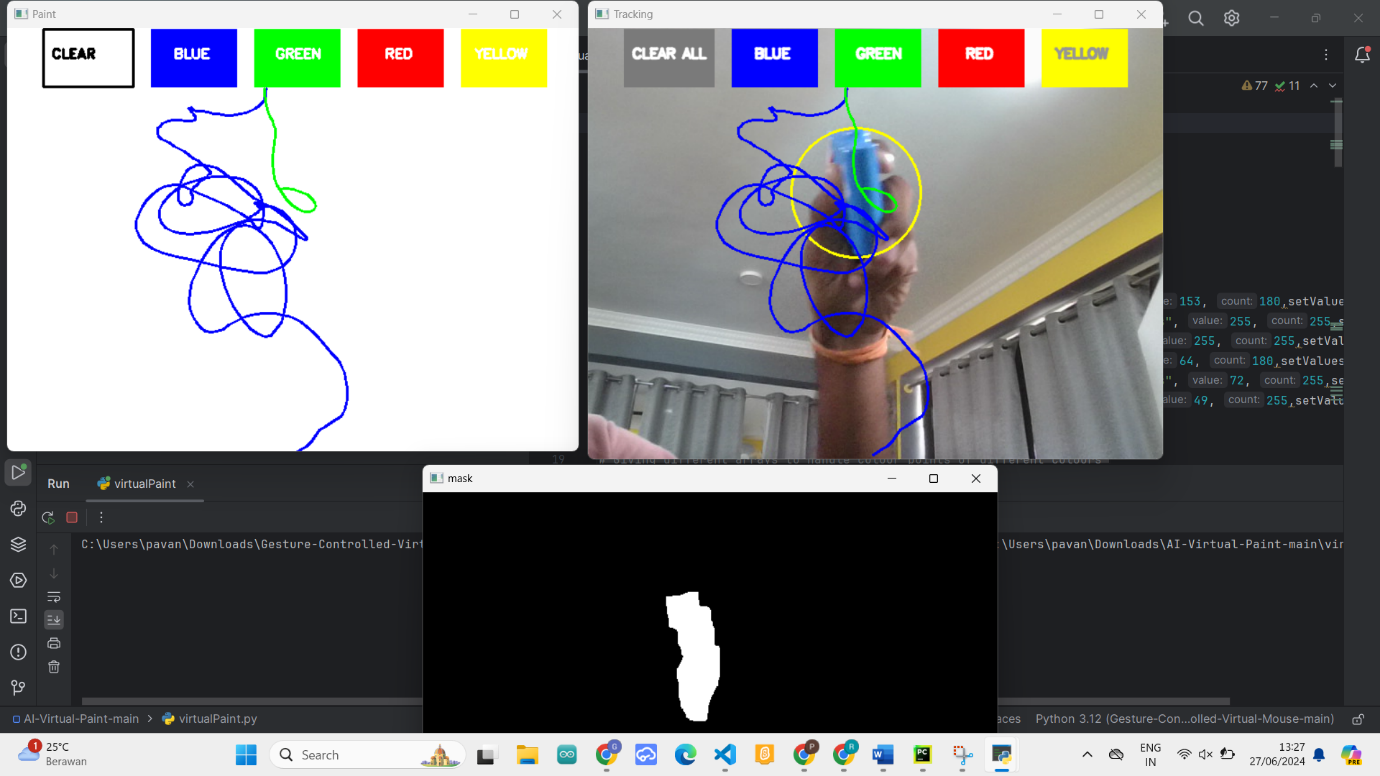
**Running:**

* Run the Virtual Mouse code

**Output:**

A screenshot of a computer screen

Description automatically generatedA screenshot of a computer

Description automatically generated

**Conclusion:**

The "Virtual Paint" project successfully demonstrates the use of AI and computer vision to create an intuitive, touchless drawing application. By tracking a colored marker or fingertip via a webcam, users can draw on a digital canvas in real-time. This project highlights the potential of computer vision for enhancing user interaction and creativity. The implementation of color detection, contour tracking, and gesture recognition ensures a seamless user experience. Future enhancements could include advanced object detection and gesture recognition, making the technology more robust and versatile. The project showcases how AI can transform traditional interactions into innovative digital experiences.