**MINI PROJECT CODES**

**HandTrackingModule:**

import cv2  
import mediapipe as mp  
import time  
  
class handDetector:  
 def \_\_init\_\_(self, mode = False, maxHands = 2, detectionCon = 0.5, trackCon = 0.5):  
 self.mode = mode  
 self.maxHands = maxHands  
 self.detectionCon = detectionCon  
 self.trackCon = trackCon  
  
 self.mpHands = mp.solutions.hands  
 self.hands = self.mpHands.Hands(self.mode, self.maxHands, self.detectionCon,self.trackCon)  
 self.mpDraw = mp.solutions.drawing\_utils  
  
 self.tipIds = [4, 8, 12, 16, 20]  
  
 def findHands(self, img, draw=True ):  
 imgRGB = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
 self.results = self.hands.process(imgRGB)  
 if self.results.multi\_hand\_landmarks:  
 for handLms in self.results.multi\_hand\_landmarks:  
 if draw:  
 self.mpDraw.draw\_landmarks(img, handLms, self.mpHands.HAND\_CONNECTIONS)  
 return img  
  
 def findPosition(self, img, handNo=0, draw=True):  
 self.lmList =[]  
 if self.results.multi\_hand\_landmarks:  
 myHand = self.results.multi\_hand\_landmarks[handNo]  
 for id, lm in enumerate(myHand.landmark):  
 h, w, c = img.shape  
 cx, cy = int(lm.x\*w), int(lm.y\*h)  
 self.lmList.append([id, cx, cy])  
 if draw:  
 cv2.circle(img, (cx, cy), 10, (255, 0, 255), cv2.FILLED)  
 return self.lmList  
  
 def fingersUp(self):  
 fingers = []  
  
 # Thumb  
 if self.lmList[self.tipIds[0]][1] < self.lmList[self.tipIds[0] - 1][1]:  
 fingers.append(1)  
 else:  
 fingers.append(0)  
  
 # 4 Fingers  
 for id in range(1, 5):  
 if self.lmList[self.tipIds[id]][2] < self.lmList[self.tipIds[id] - 2][2]:  
 fingers.append(1)  
 else:  
 fingers.append(0)  
 return fingers  
  
def main():  
 cap = cv2.VideoCapture(0)  
 pTime = 0  
 cTime = 0  
 detector = handDetector()  
  
 while True:  
 success, img = cap.read()  
 img = detector.findHands(img)  
 lmList = detector.findPosition(img)  
 if len(lmList) != 0:  
 print(lmList[8])  
  
 cTime = time.time()  
 fps = 1 / (cTime - pTime)  
 pTime = cTime  
  
 cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT\_HERSHEY\_PLAIN, 3, (255, 0, 255), 3)  
  
 cv2.imshow("Image", img)  
 cv2.waitKey(1)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**VirtualPainter:**

import cv2  
import numpy as np  
import time  
import os  
import HandTrackingModule as htm  
  
##########################  
brushThickness = 15  
eraserThickness = 50  
##########################  
  
folderPath = "Header"  
myList = os.listdir(folderPath)  
print(myList)  
overLayList = []  
for imPath in myList:  
 image = cv2.imread(f'{folderPath}/{imPath}')  
 overLayList.append(image)  
print(len(overLayList))  
header = overLayList[0]  
drawColor = (255, 255, 0)  
  
cap = cv2.VideoCapture(0)  
cap.set(3, 1280)  
cap.set(4, 720)  
  
detector = htm.handDetector(detectionCon=0.85)  
xp, yp = 0, 0  
imgCanvas = np.zeros((720, 1280, 3), np.uint8)  
  
while True:  
  
 # 1. Import image  
 success, img = cap.read()  
 img = cv2.flip(img, 1)  
  
 # 2. Find Hand Landmarks  
 img = detector.findHands(img)  
 lmList = detector.findPosition(img, draw=False)  
  
 if len(lmList) != 0:  
 #print(lmList)  
  
 # Tip of index and middle finger  
 x1, y1 = lmList[8][1:]  
 x2, y2 = lmList[12][1:]  
  
 # 3. Check which fingers are up  
  
 fingers = detector.fingersUp()  
 #print(fingers)  
  
 # 4. If Selection Mode - Two fingers are up  
 if fingers[1] and fingers[2]:  
 xp, yp = 0, 0  
 print("Selection Mode")  
 # Checking for the Click  
 if y1 < 130:  
 if 250 < x1 < 450:  
 header = overLayList[0]  
 drawColor = (255, 255, 0)  
 elif 550 < x1 < 750:  
 header = overLayList[1]  
 drawColor = (0, 0, 255)  
 elif 800 < x1 < 950:  
 header = overLayList[2]  
 drawColor = (0, 255, 0)  
 elif 1050 < x1 < 1200:  
 header = overLayList[3]  
 drawColor = (0, 0, 0)  
 cv2.rectangle(img, (x1, y1 - 25), (x2, y2 + 25), drawColor, cv2.FILLED)  
  
 # 5. If Drawing Mode -Index finger is up  
 if fingers[1] and fingers[2] == False:  
 cv2.circle(img, (x1, y1), 15, drawColor, cv2.FILLED)  
 print("Drawing Mode")  
 if xp == 0 and yp == 0:  
 xp, yp = x1, y1  
  
 if drawColor == (0, 0, 0):  
 cv2.line(img, (xp, yp), (x1, y1), drawColor, eraserThickness)  
 cv2.line(imgCanvas, (xp, yp), (x1, y1), drawColor, eraserThickness)  
 else:  
 cv2.line(img, (xp, yp), (x1, y1), drawColor, brushThickness)  
 cv2.line(imgCanvas, (xp, yp), (x1, y1), drawColor, brushThickness)  
  
 xp, yp = x1, y1  
  
 imgGray = cv2.cvtColor(imgCanvas, cv2.COLOR\_BGR2GRAY)  
 \_, imgInv = cv2.threshold(imgGray, 50, 255, cv2.THRESH\_BINARY\_INV)  
 imgInv = cv2.cvtColor(imgInv, cv2.COLOR\_GRAY2BGR)  
 img = cv2.bitwise\_and(img, imgInv)  
 img = cv2.bitwise\_or(img, imgCanvas)  
  
 # Setting the header image  
 img[0:130, 0:1280] = header  
 img = cv2.addWeighted(img, 0.5, imgCanvas, 0.5, 0)  
 cv2.imshow("Image", img)  
 cv2.imshow("Canvas", imgCanvas)  
 #cv2.imshow("Inv", imgInv)  
 cv2.waitKey(1)