SketchMotion

This project will guide you through creating a camera-controlled drawing application. You'll learn how to use computer vision and hand tracking to translate your hand movements into digital art.

BY

Chinni Vinay Kumar (225890510/57)

Jayanth (225890282/26)

Donthabhaktuni Chaitanya (225890444/50)

Introduction to Computer Vision and Hand Tracking

1 Computer Vision

Computer vision enables computers to "see" and interpret images and videos.

2 Hand Tracking

Hand tracking algorithms identify and locate the hand in real-time, using camera input.

3 Applications

Hand tracking has numerous applications, from virtual reality and gaming to medical imaging and robotics.

Enabling Camera Access on the PC

1

Permissions

Request camera access from the user.

2

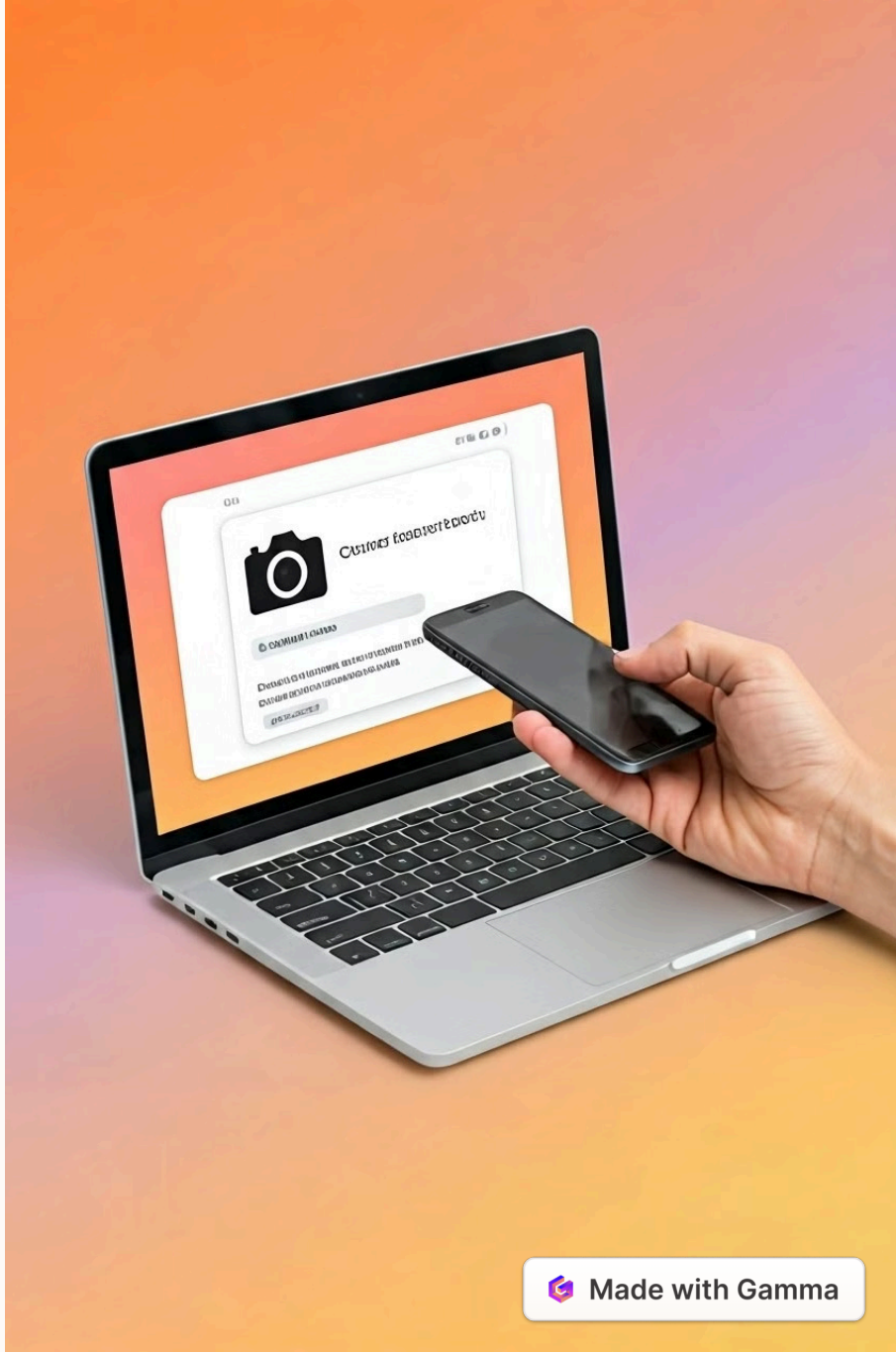
Libraries

Use appropriate libraries like OpenCV or MediaPipe for camera access and processing.

3

Input Capture

Capture video frames from the camera.



Implementing Hand Tracking Algorithms

OpenCV

OpenCV offers hand tracking algorithms based on traditional image processing techniques.

MediaPipe

MediaPipe provides machine learning-based hand tracking solutions, offering higher accuracy and performance.

Mapping Hand Movements to Screen Coordinates

Keypoints

Extract keypoints like finger tips and wrist from the tracked hand.

1

Mapping

Map the hand's coordinates to corresponding screen coordinates.

3

2

Calibration

Calibrate the hand position relative to the screen.

Selecting Colors Using Hand Gestures

Circular Motion

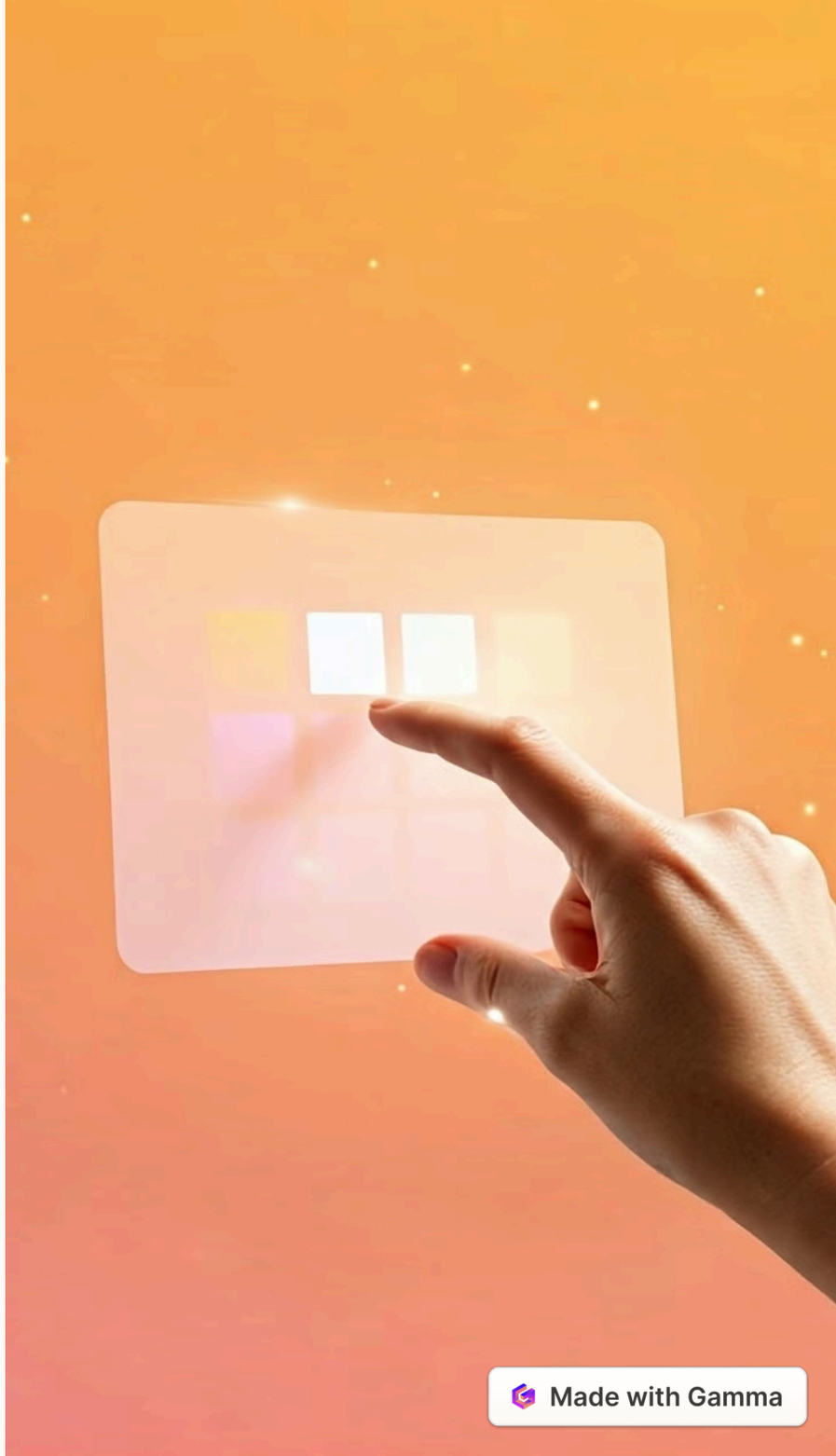
Rotating the hand in a circle might select a color from a virtual palette.

Finger Pointing

Pointing a finger towards a color on a palette could select it.

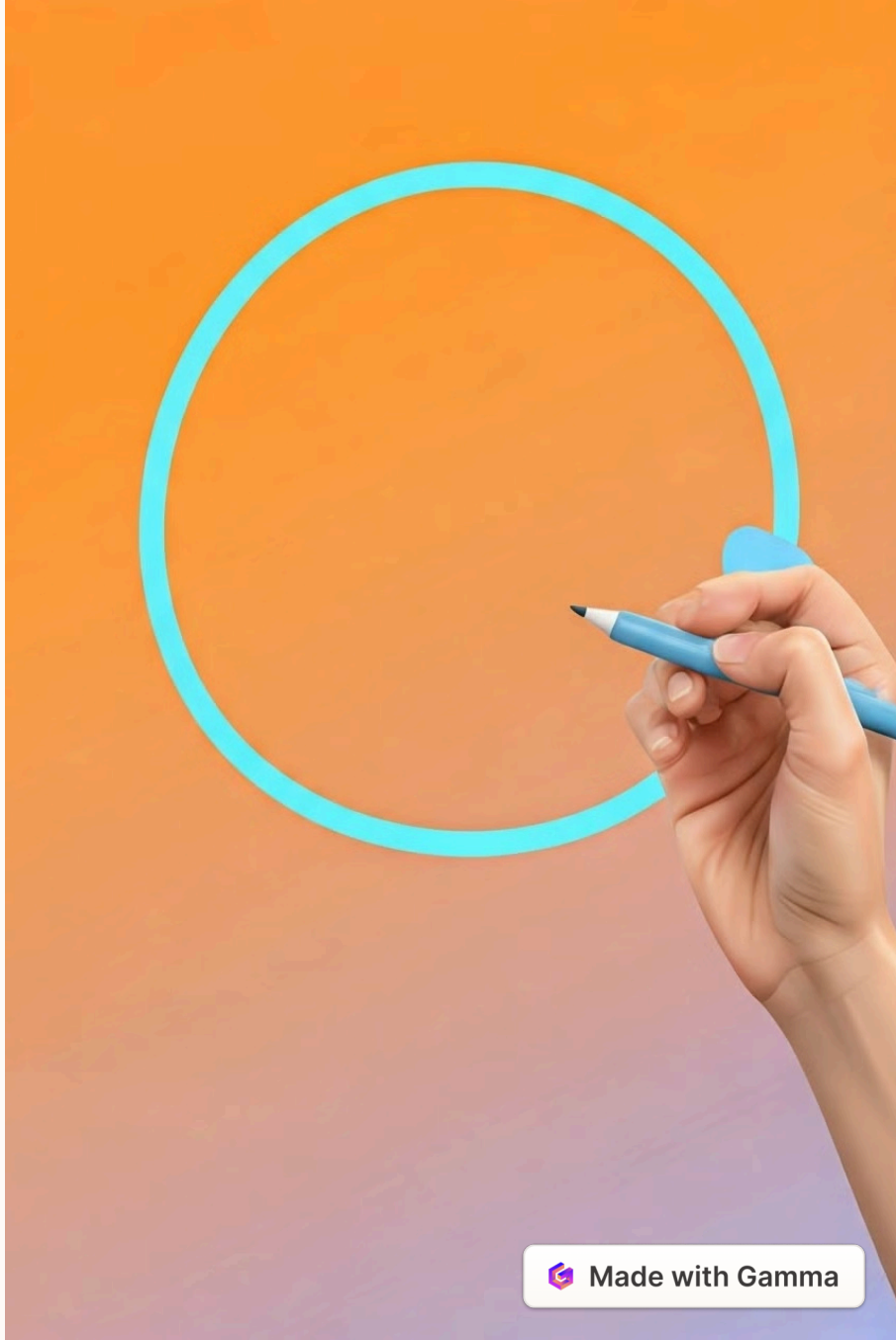
Hand Position

The position of the hand over the screen might correspond to a gradient of colors.



Drawing Shapes Based on Hand Movements

Movement	Shape
Swiping	Line
Circular Motion	Circle
Pinch	Square



Real-Time Rendering and Visualization

Canvas

Use a canvas to display the drawings in real time.

Updating

Continuously update the canvas with the new shapes and colors based on the hand movements.

Smoothness

Use smoothing algorithms to make the drawing appear smooth and natural.

Potential Applications and Future Enhancements

Educational Games

Interactive learning apps for children.

Virtual Reality

Create immersive and intuitive VR experiences.

Code For handtracking

```
import cv2
import mediapipe as mp
import math

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(static_image_mode=self.mode,
max_num_hands=self.maxHands,
min_detection_confidence=int(self.detectionCon  100) / 100.0,
min_tracking_confidence=int(self.trackCon  100) / 100.0)
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):
xList = []
yList = []
bbox = []
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)
xList.append(cx)
yList.append(cy)
self.lmList.append([id, cx, cy])
if draw:
cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)

if xList and yList:
xmin, xmax = min(xList), max(xList)
ymin, ymax = min(yList), max(yList)
bbox = xmin, ymin, xmax, ymax

return self.lmList

def fingersUp(self):
fingers = []
if self.lmList:
if self.lmList[self.tipIds[0]][1] < self.lmList[self.tipIds[0] - 1][1]:
fingers.append(1)
else:
fingers.append(0)

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 findDistance(self, p1, p2, img, draw=True, r=15, t=3):
x1, y1 = self.lmList[p1][1:]
x2, y2 = self.lmList[p2][1:]
cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

 if draw:
cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), t)
cv2.circle(img, (x1, y1), r, (255, 0, 255), cv2.FILLED)
cv2.circle(img, (x2, y2), r, (255, 0, 255), cv2.FILLED)
cv2.circle(img, (cx, cy), r, (0, 0, 255), cv2.FILLED)

length = math.hypot(x2 - x1, y2 - y1)
return length, img, [x1, y1, x2, y2, cx, cy]


def main():
import time
pTime = 0
 cTime = 0
 cap = cv2.VideoCapture(0)
detector = handDetector()
while True:
success, img = cap.read()
img = detector.findHands(img)
lmList = detector.findPosition(img)
if lmList:
print(lmList[4])

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()
```

Code for Virtual printer

```
import cv2

 import numpy as np

 import os import HandTrackingModule as htm

folderPath = "Header"

myList = os.listdir(folderPath) overlayList = []

for imPath in myList: image = cv2.imread(f'{folderPath}/{imPath}')

image = cv2.resize(image, (1280, 125)) # Resize the image to fit the header space

header = overlayList[0]

 drawColor = (255, 0, 255)

 brushThickness = 15

 eraserThickness = 100

 xp, yp = 0, 0 imgCanvas = np.zeros((720, 1280, 3), np.uint8)

currentShape = None

shapeStart = (0, 0)

shapeEnd = (0, 0)

shapeDrawing = False

def virtual_Painter():

cap = cv2.VideoCapture(0) cap.set(3, 1280) cap.set(4, 720)



detector = htm.handDetector(detectionCon=0.85, maxHands=1)
global xp, yp, imgCanvas, header, drawColor, brushThickness, currentShape, shapeStart, shapeEnd, shapeDrawing

while True:
    success, img = cap.read()
    img = cv2.flip(img, 1)
    img = detector.findHands(img)
    lmList = detector.findPosition(img, draw=False)

    if len(lmList) != 0:
        # Tip positions
        x1, y1 = lmList[8][1:]
        x2, y2 = lmList[12][1:]
        x0, y0 = lmList[4][1:]

        fingers = detector.fingersUp()

        # Selection mode - Two fingers up
        if fingers[1] and fingers[2]:
            xp, yp = 0, 0
            shapeDrawing = False

            # Checking for the click on header
            if y1 < 125:
                if 250 < x1 < 450:
                    header = overlayList[0]
                    drawColor = (255, 0, 255)
                elif 550 < x1 < 750:
                    header = overlayList[1]
                    drawColor = (255, 0, 0)
                elif 800 < x1 < 950:
                    header = overlayList[2]
                    drawColor = (0, 255, 0)
                elif 1050 < x1 < 1200:
                    header = overlayList[3]
                    drawColor = (0, 0, 0)

            # Checking for shape selection
            elif 125 < y1 < 210:
                if 50 < x1 < 200:
                    currentShape = "rectangle"
                elif 250 < x1 < 400:
                    currentShape = "circle"

            cv2.rectangle(img, (x1, y1 - 25), (x2, y2 + 25), drawColor, cv2.FILLED)

        # Drawing mode - Index finger up
        elif fingers[1] and not fingers[2]:
            if currentShape:
                if not shapeDrawing:
                    shapeStart = (x1, y1)
                    shapeDrawing = True
                shapeEnd = (x1, y1)

            else:
                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

        # Shape drawing logic
        if shapeDrawing:
            if currentShape == "rectangle":
                cv2.rectangle(img, shapeStart, shapeEnd, drawColor, brushThickness)
                cv2.rectangle(imgCanvas, shapeStart, shapeEnd, drawColor, brushThickness)
            elif currentShape == "circle":
                radius = int(((shapeEnd[0] - shapeStart[0]) ** 2 + (shapeEnd[1] - shapeStart[1]) ** 2) ** 0.5 / 2)
                center = ((shapeStart[0] + shapeEnd[0]) // 2, (shapeStart[1] + shapeEnd[1]) // 2)
                cv2.circle(img, center, radius, drawColor, brushThickness)
                cv2.circle(imgCanvas, center, radius, drawColor, brushThickness)

    # Merge the canvas and video feed
    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[0:125, 0:1280] = header

    cv2.imshow("Image", img)
    cv2.imshow("Canvas", imgCanvas)
    cv2.waitKey(1)
```

if **name** == **"main"**: print("Starting Virtual Painter") virtual_Painter()

THANK YOU

Hope you dont have any Questions

NOTE : NO QUESTIONS ARE ENTERTAINED