Electronics Club, IIT Guwahati

Project: Hand Gesture

Mentored by: Harsh Vardhan Singh & Rahul Aggarwal

Link: https://github.com/elecclubiitg/Hang-Gesture-2021

Report Prepared by:

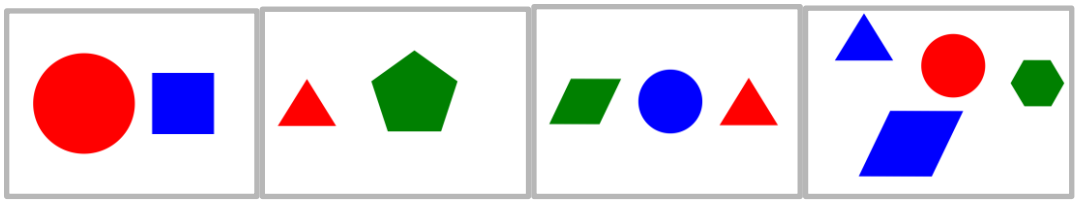
1. Amish Agrawal
2. Arpit Sureka
3. Harsha Vardhan Lomisa
4. Kesava Karthik Illa

Task 1: Opencv Assignment

The first task was to recognize objects in the provided photos and provide information about them, such as shape, color, center coordinates, and area.

We utilized Numpy in addition to the OpenCV package.

To understand this task, we first need to know how an image is stored. Each pixel carries three values, each corresponding to different channels, i.e., RGB. These values range from 0-255 (according to the intensity of color in that channel). In a computer, this is stored in the form of a 3D matrix.



```
shapes = {}

def scan_image(img_file_path):

    global shapes
    shapes.clear()

    image = cv2.imread(img_file_path)
    img_grey = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(img_grey, 150, 255, cv2.THRESH_BINARY)

    contours, hierarchy = cv2.findContours(image=thresh, mode=cv2.RETR_TREE, method=cv2.CHAIN_APPROX_SIMPLE)

    for contour in contours:
        approx = cv2.approxPolyDP(contour, 0.01 * cv2.arcLength(contour, True), True)
        area = cv2.contourArea(contour)
        if contour[0][0][0] == 0:
            continue

        M = cv2.moments(contour)

        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
```

```
        if image[cy,cx][0] >  50:
            colour = "Blue"
        elif image[cy,cx][1] > 50:
            colour = "Green"
        elif image[cy,cx][2] > 50:
            colour = "Red"

        if len(approx) == 3:
            shapes["triangle"] = [colour,area,cx,cy]
        elif len(approx) == 4:
            shapes["quadrilateral"] = [colour,area,cx,cy]
        elif len(approx) == 5:
            shapes["pentagon"] = [colour,area,cx,cy]
        elif len(approx) == 6:
            shapes["hexagon"] = [colour,area,cx,cy]
        else:
            shapes["circle"] = [colour,area,cx,cy]

    cv2.destroyAllWindows()


    return shapes
```

Explanation:

- First of all, image preprocessing is being done so that the boundaries of the shapes can be easily detected. For this preprocessing we converted the colored image to a black and white image and then changed all the pixels with intensity less than 150 to a black pixel that is 0 and greater than 150 to white pixel i.e 255.
- This preprocessed image is passed to find contours i.e boundaries in the given image. Contours basically work by calculating the change in values that come when we move from one pixel to its adjacent pixel if there is sudden and a big change in the value it interprets it as a boundary.
- These contours can be then used to calculate details such as area, shape and centroid.
- Since we have the centroid of the shape we can find the intensity of each channel at the centroid and use “if/else” statement to know the color of the shape at that point.

Task 2: Mediapipe Assignment

1. Libraries used

```
import cv2
import mediapipe as mp
import time
```

1. cv2 - OpenCV
2. mediapipe - an open-source cross-platform framework with ML solutions for live and streaming media.
3. time - FPS counter

2 . class and constructor

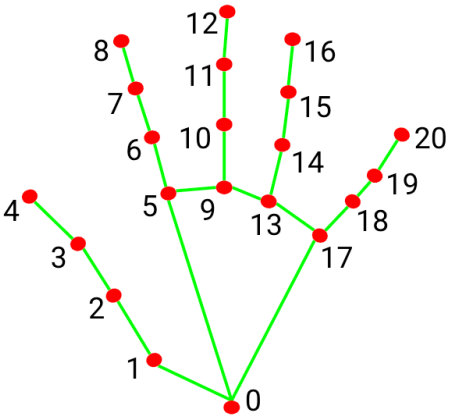
```
class handDetector():

    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]
```

- The main code for us is to make the class handDetector with all the required functions for getting the info from the detected hand.
- Firstly the initialize constructor is already defined.So, if needed we can make any changes to this constructor.
- In the constructor we use different parameters which are :
    1. mode : If set to `false`, the solution treats the input images as a video stream.
    2. maxHands: Maximum number of hands to detect
    3. detectionCon: Minimum confidence value (`[0.0, 1.0]`) from the hand detection model for the detection to be considered successful.
    4. trackCon:Minimum confidence value (`[0.0, 1.0]`) from the landmark-tracking model for the hand landmarks to be considered tracked successfully, or otherwise hand detection will be invoked automatically on the next input image.
    5. mp.solutions.hands: Mediapipe Hands solutions.

3 . Findhands

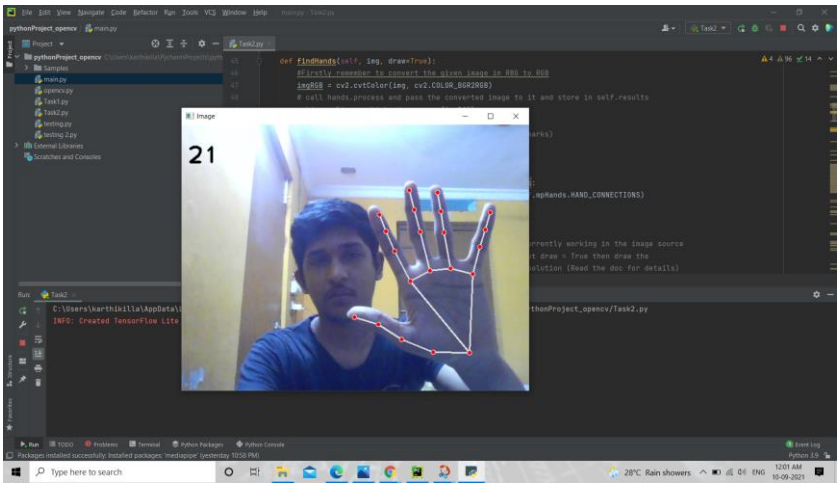
```
def findHands(self, img, draw=True):
    imgrgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    self.results = self.hands.process(imgrgb)
    if draw:
        if self.results.multi_hand_landmarks:
            for hans in self.results.multi_hand_landmarks:
                self.mpDraw.draw_landmarks(img, hans, self.mpHands.HAND_CONNECTIONS)

    return img
```

- findHands function takes the image source from the calling block and then draws the hands with all the landmarks of 21 different points on our hand using mediapipe Hands solution.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

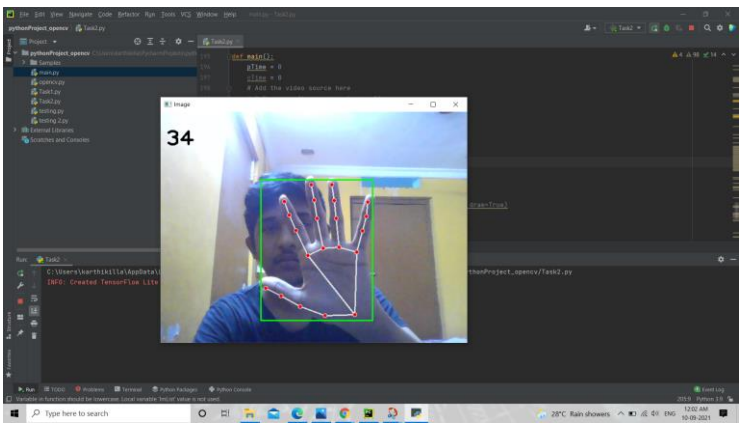
- Firstly in this function, we convert the image from BGR to RGB format.
- Then we call hands.process and pass the RGB image which processes the image and gets landmarks for us and stores it in self.results.
- Lastly we draw the landmarks on the image and return it.



4 . Find position

```
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):
            #print(id, lm)
            h, w, c = img.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            xList.append(cx)
            yList.append(cy)
            #print(id, cx, cy)
            self.lmList.append([id, cx, cy])
            cv2.circle(img, (cx, cy), 5, (255, 0, 0), cv2.FILLED)

        xmin, xmax = min(xList), max(xList)
        ymin, ymax = min(yList), max(yList)
        bbox = [(xmin, ymin),(xmax, ymax)]
        if draw:
            cv2.rectangle(img, (xmin - 20, ymin - 20), (xmax + 20, ymax + 20), (0, 0, 255), 2)

    return self.lmList, bbox
```

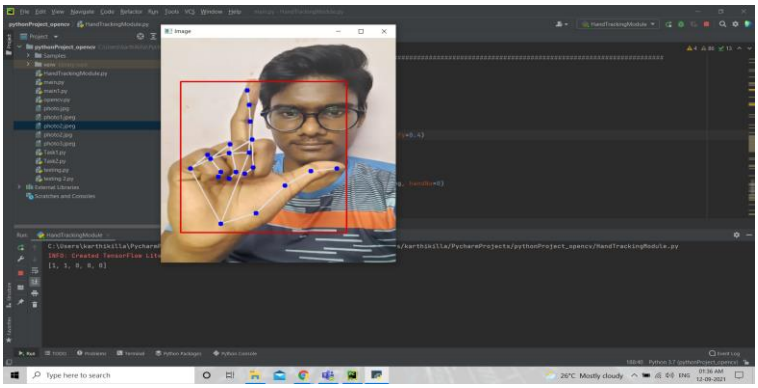
- findPosition function takes the image source, and we are currently working in the image source and then draw the hand positions with all the landmarks using mediapipe Hands solution.
- In the findPosition Function, we basically find the coordinates of all the 21 landmarks on our hand and store them in a landmark list for future calculations.
- Mediapipe gives the coordinates which are normalized to 0.0 and 1.0
- So, we convert them to exact coordinates and store them in the list.
- Finally, we draw a rectangle around the hand and the landmarks are marked with circles.



5 . Finger up

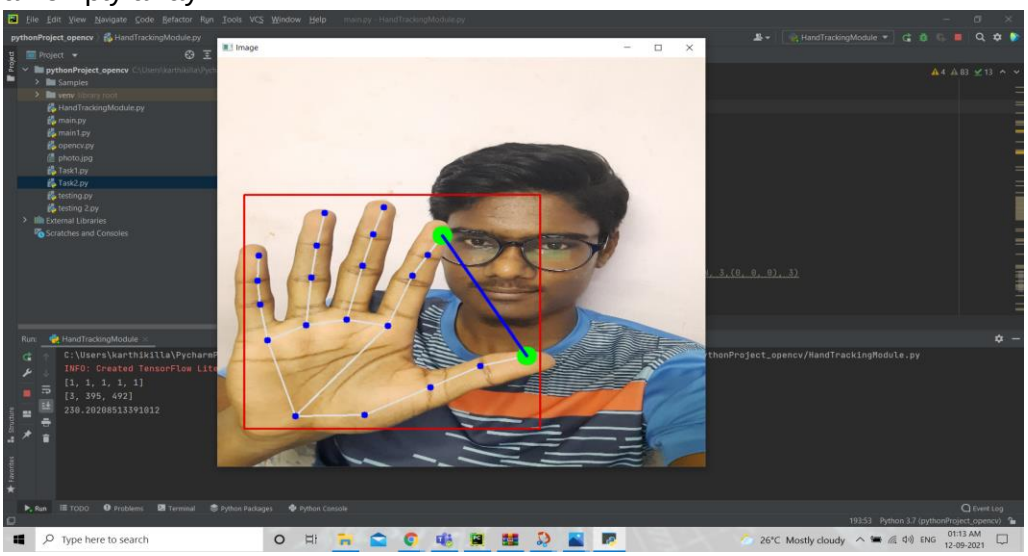
```
def fingersUp(self):
    fingers = []
    rotlmlist = []
    if self.lmList:
        x = self.lmList[9][1]-self.lmList[0][1]
        y = self.lmList[9][2]-self.lmList[0][2]
        cos = y / ((x ** 2 + y ** 2) ** 0.5)
        sin = x / ((x ** 2 + y ** 2) ** 0.5)
        for i in range(21):
            X = self.lmList[i][1] * cos - self.lmList[i][2] * sin
            Y = self.lmList[i][1] * sin + self.lmList[i][2] * cos
            rotlmlist.append([i, X, Y])
        if rotlmlist[17][1] > rotlmlist[4][1]:
            if rotlmlist[4][1] >= rotlmlist[3][1]:
                fingers.append(0)
            else:
                fingers.append(1)
        if rotlmlist[4][1] > rotlmlist[17][1]:
            if rotlmlist[3][1] > rotlmlist[4][1]:
                fingers.append(0)
            else:
                fingers.append(1)
        for i in range(1,5):
            if rotlmlist[self.tipIds[i]-2][2] > rotlmlist[self.tipIds[i]][2]:
                fingers.append(0)
            else:
                fingers.append(1)

    return fingers
```

- The function fingerUp tells us about which fingers are not folded by returning an array of 5 elements containing combinations of 0's and 1's.
- First element of array represents status of thumb finger, second element represents status of index finger and so on will continue till fifth element.
- The 0 is given as the element if the finger is closed and 1 for not closed.
- For example: in a case where if the thumb and the index finger are open and all other fingers are closed, then the function will give [1,1,0,0,0].
- In this, first we will rotate the axis system to get hand upright position coordinates.
- Then we will compare tips of the fingers with other points to determine if it is in folded state or not and assign 0 or 1 to respective elements.
- If no hand is detected, then it will return a empty array.



6 . Find distance

```
def findDistance(self, p1, p2, img, draw=True,r=15, t=3):

    if self.lmList == []:
        length = 0
        return length, img, []
    x1, y1 = self.lmList[p1][1:]
    x2, y2 = self.lmList[p2][1:]
    cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
    length = ((x1-x2) ** 2 + (y1-y2) ** 2) ** 0.5
    if draw:
        cv2.circle(img, (x1, y1), r, (0, 255, 0), -1)
        cv2.circle(img, (x2, y2), r, (0, 255, 0), -1)
        cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), t)


    return length, img, [x1, y1, x2, y2, cx, cy]
```

- The function findDistance will give the distance between two points and coordinates of endpoints and midpoint of the line segment formed by joining.
- The Id's of the points are passed as input. If draw is given to be true then the function will highlight the points and will draw a line segment joining them.
- It will find the length, midpoints by using coordinates of the points.
- If no hand is detected then it will return length to be 0 and coordinates to be an empty array.
- 

TASK 3 - AUTOPY

1. Libraries used

```
import HandTrackingModule as htm
import cv2
import numpy as np
import autopy
import mouse
import time
```

- **cv2** - openCV for live video capture
- **autopy and mouse** - controlling the mouse movements
- **HandTrackingModule** - mediapipe hand tracking module completed in task 2
- **time** - to implement FPS counter

2. Initializing Variables

```
wCam, hCam = 1280, 720
frameR = 40
smoothening = 2

pTime = 0
plocX, plocY = 0, 0
clocX, clocY = 0, 0

cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
detector = htm.handDetector(maxHands=1)
wScr, hScr = autopy.screen.size()
```

3.Main code loop

```
while True:
    _, img = cap.read()
    # Find Hand LandMarks
    img = detector.findHands(img)
    lmList, bbox = detector.findPosition(img)
    # Get tip of index and middle fingers
    if len(lmList) != 0:
        x1, y1 = lmList[8][1:]
        x2, y2 = lmList[12][1:]

        # Check which finger is up
        fingers = detector.fingersUp()
        print(fingers)
        cv2.rectangle(img, (frameR, frameR), (wCam-frameR, hCam-frameR), (255,0,255),2)
        # Only index finger : moving mode
        if fingers[1] == 1 and fingers[2] == 0 and fingers[3] == 0 and fingers[4] == 0:
            # Convert Coordinates
            x3 = np.interp(x1, (frameR, wCam - frameR), (0, wScr))
            y3 = np.interp(y1, (frameR, hCam - frameR), (0, hScr))
            # Smoothen Values
            clocX = plocX + (x3 - plocX)/smoothening
            clocY = plocY + (y3 - plocY) / smoothening
            # Move Mouse
            autopy.mouse.move(wScr - clocX, clocY)
            cv2.circle(img, (int(x1), int(y1)), 15, (0,255,0), cv2.FILLED)
            plocX, plocY = clocX, clocY
        if fingers[1] == 1 and fingers[2] == 1:
            length, img, lineInfo = detector.findDistance(8, 12, img, draw=False)
            if fingers[3] == 1 and fingers[4] == 0:
```

```
        if fingers[3] == 1 and fingers[4] == 0:
            mouse.click('right')
            cv2.putText(img, "Right Click", (15, 40), 1, 1, (255, 0, 0), 3)
        if length < 65:
            cv2.circle(img, (lineInfo[4], lineInfo[5]), 10, (0, 255, 0), -1)
            mouse.click('left')
            cv2.putText(img, "Left Click", (15, 40), 1, 1, (255, 0, 0), 3)
    if fingers[0] == 1 and fingers[1] == 1:
        length, _, _ = detector.findDistance(8, 4, img, draw=False)
        if length < 50:
            autopy.mouse.toggle(down=True)
        if length > 50:
            autopy.mouse.toggle(down=False)
    if fingers[1] == 0 and fingers[2] == 0 and fingers[3] == 0 and fingers[4] == 0:
        mouse.wheel(-1)
        cv2.putText(img, "Scroll Down", (15, 40), 1, 1, (255, 0, 0), 3)
    elif fingers[1] == 1 and fingers[2] == 1 and fingers[3] == 1 and fingers[4] == 1:
        mouse.wheel(1)
        cv2.putText(img, "Scroll Up", (15, 40), 1, 1, (255, 0, 0), 3)
```

```
    cTime = time.time()
    fps = 1/(cTime - pTime)
    pTime = cTime
    cv2.putText(img, str(int(fps)), (15, 15), 1, 1, (255, 0, 0), 3)
    cv2.resize(img, (640, 360))
    cv2.imshow("Image", img)
    if cv2.waitKey(5) & 0xFF == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Explanation:

- Using the information from the hand tracking module about the position of the fingers(fingertips), we have implemented some functions using if-else conditions.
- Only index finger up - mouse traversing
  - When it is brought down below its midpoint - scroll down
  - While it is up, if the other 3 fingers are put up - scroll up
- Index and middle finger up -
  - No mouse movements
  - When the distance between the tips of both fingers becomes less than a certain threshold value - mouse left click
  - While the 2 are up, if the 3rd finger is put up - mouse right click
- Index finger and thumb up -
  - When the distance between their tips becomes less than a certain threshold value - mouse left click toggle
  - This is used for drag and drop functions.

Screenshot of Working Project: