CV Project

Real-Time Motion Detection and Facial Feature Tracking using YOLO, Haar Cascades, and Optical Flow

By Gurbaksh Lal (2023PCS2029)

Using requirement or tools for project:

- Motion Detection.ipynb
- coco.names
- Computer Vision work.ppt
- FrameInMotion_time.csv
- Motion Detection.py
- Poster.pdf
- yolov3.cfg
- yolov3.weights

Red: Required file to run the project here we use coco and yolo

Yellow: Programming files or main file of project

Green: Output CSV file capture frame movement time

Code:

Python->

```
import cv2

import pandas as pd

from datetime import datetime

import numpy as np


# Initialize the DataFrame with start and end time

df = pd.DataFrame(columns=["Start", "End"])

motionImage = []

time = []

stillImage = None


# Load pre-trained Haar cascades for face, eyes, and spectacles detection

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')

spectacles_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_eye_tree_eyeglasses.xml')


# Load YOLO model for object detection

net = cv2.dnn.readNet(r'C:\Users\gurba\Project\Computer Vision Project\yolov3.weights',
r'C:\Users\gurba\Project\Computer Vision Project\yolov3.cfg')

with open(r'C:\Users\gurba\Project\Computer Vision Project\coco.names', 'r') as f:

    classes = [line.strip() for line in f.readlines()]


layer_names = net.getLayerNames()

output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]


# Parameters for Lucas-Kanade optical flow
```

```
lk_params = dict(winSize=(15, 15), maxLevel=2, criteria=(cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Create some random colors
color = np.random.randint(0, 255, (100, 3))

# Capturing video
video = cv2.VideoCapture(0)
video.set(cv2.CAP_PROP_FPS, 30)  # Set frame rate to 30 FPS

# Background Subtractor
fgbg = cv2.createBackgroundSubtractorMOG2()

# Take first frame and find corners in it
ret, old_frame = video.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, maxCorners=100, qualityLevel=0.3,
minDistance=7, blockSize=7)

# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)

while True:
    # Start reading image from video
    check, frame = video.read()
    motion = 0

    # Apply background subtraction
    fgmask = fgbg.apply(frame)

    # Convert color image to gray_scale image
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
    gray = cv2.GaussianBlur(gray, (5, 5), 0)
    if stillImage is None:
        stillImage = gray
        continue
    # Still Image and current image.
    diff_frame = cv2.absdiff(stillImage, gray)

    # Change the image to white if static background and current frame is greater than 25.
    thresh_frame = cv2.threshold(diff_frame, 25, 255, cv2.THRESH_BINARY)[1]
    thresh_frame = cv2.dilate(thresh_frame, None, iterations=2)
    # Finding contour and hierarchy from a moving object.
    contours, hierachy = cv2.findContours(thresh_frame.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:
        if cv2.contourArea(contour) < 5000:  # Lower the threshold
            continue
        motion = 1
        (x, y, w, h) = cv2.boundingRect(contour)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 1)
    # Append current status of motion
    motionImage.append(motion)
    motionImage = motionImage[-2:]
    # Append Start time of motion
    if len(motionImage) >= 2 and motionImage[-1] == 1 and motionImage[-2] == 0:
        time.append(datetime.now())

    # Append End time of motion
    if len(motionImage) >= 2 and motionImage[-1] == 0 and motionImage[-2] == 1:
        time.append(datetime.now())

    # Detect faces
```

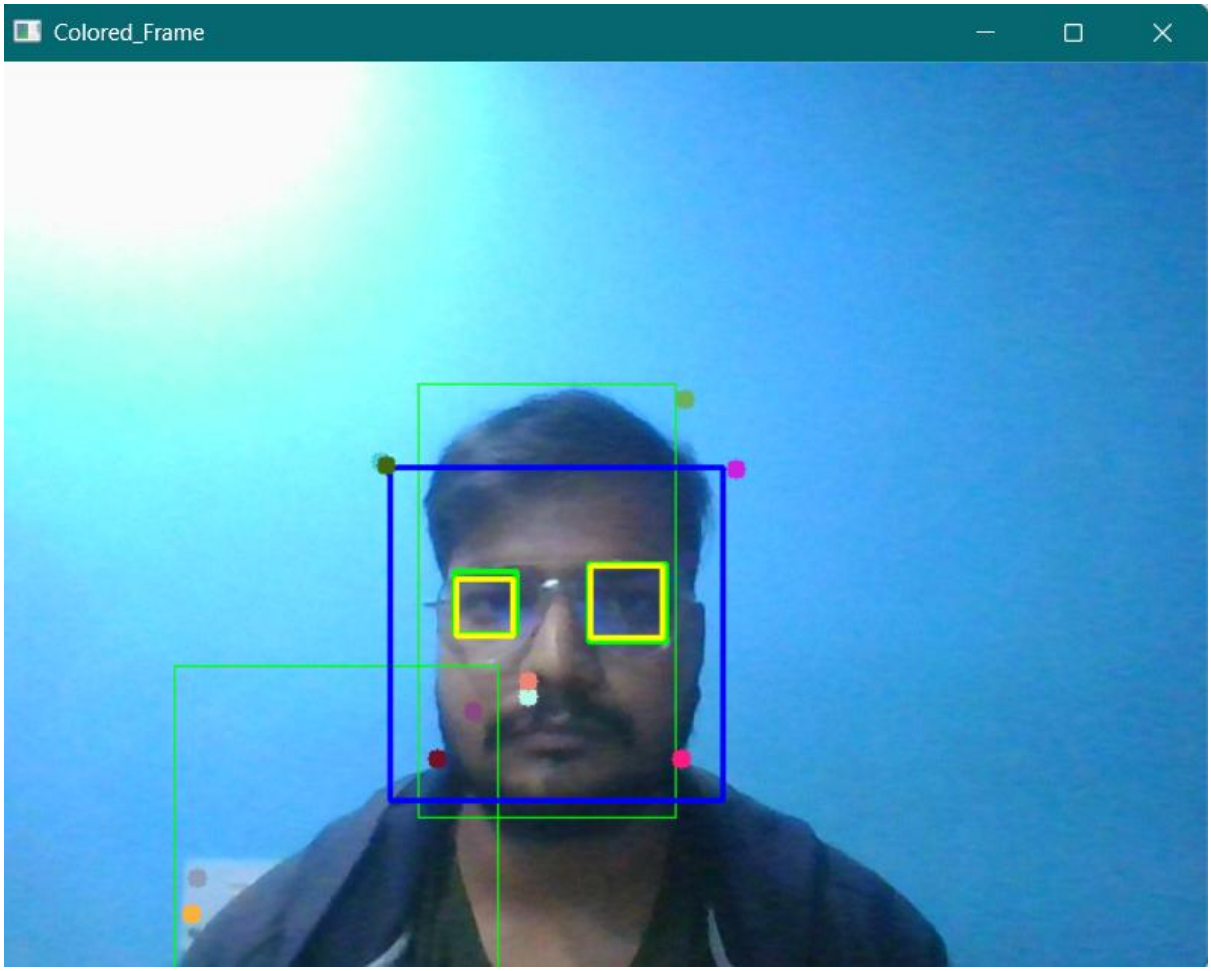
```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x, y, w, h) in faces:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
    roi_gray = gray[y:y + h, x:x + w]
    roi_color = frame[y:y + h, x:x + w]

    # Detect eyes
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex, ey, ew, eh) in eyes:
        cv2.rectangle(roi_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)

    # Detect spectacles
    spectacles = spectacles_cascade.detectMultiScale(roi_gray)
    for (sx, sy, sw, sh) in spectacles:
        cv2.rectangle(roi_color, (sx, sy), (sx + sw, sy + sh), (0, 255, 255), 2)

# Calculate optical flow
frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Ensure p0 is not None and has valid points
if p0 is not None and len(p0) > 0:
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)
    if p1 is not None:
        good_new = p1[st == 1]
        good_old = p0[st == 1]
    else:
        good_new = np.array([])
        good_old = np.array([])
else:
    good_new = np.array([])
    good_old = np.array([])
```

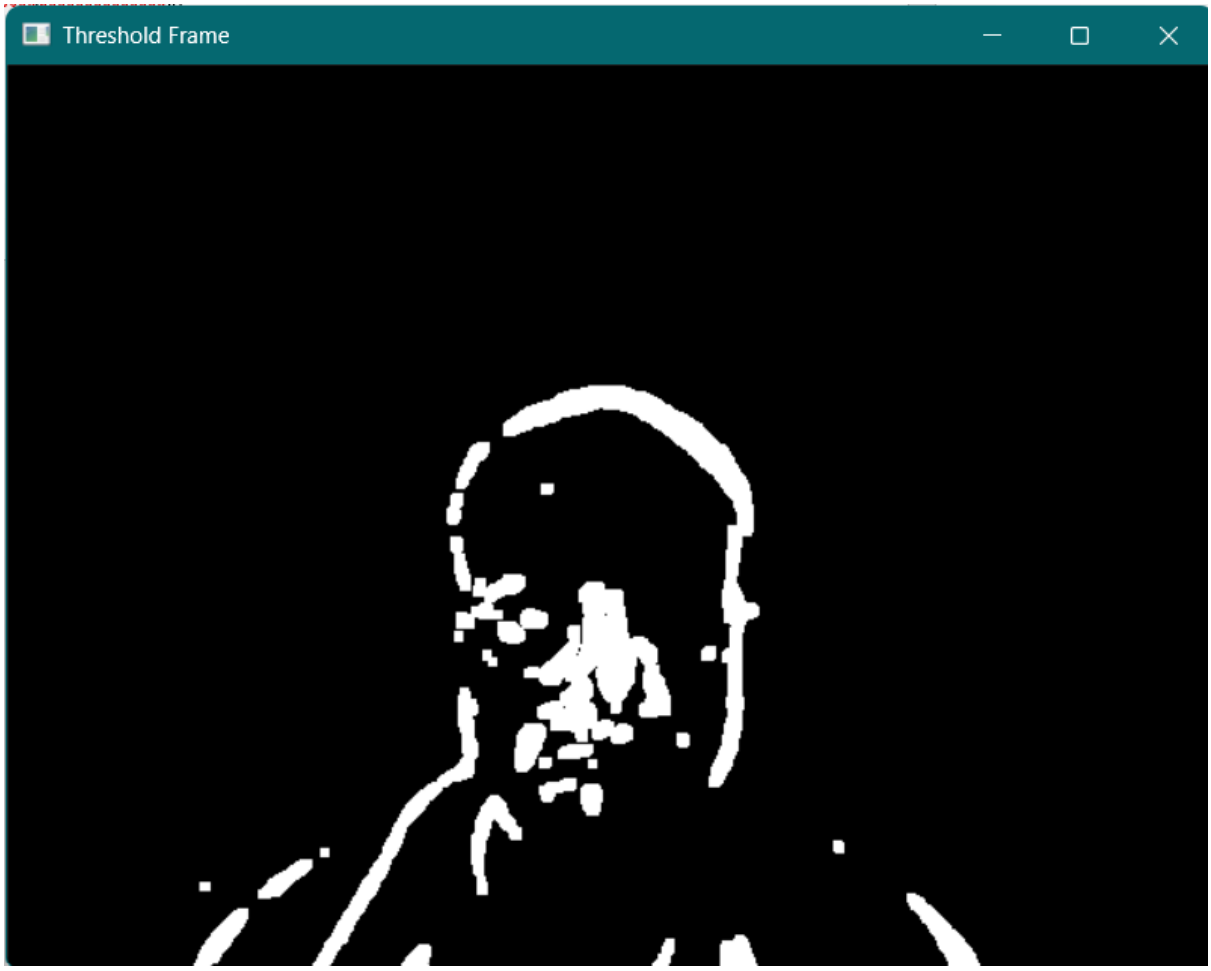
```
# Ensure good_new is a numpy array
good_new = np.array(good_new)

# Now you can reshape it
p0 = good_new.reshape(-1, 1, 2)



# Draw the tracks
for i, (new, old) in enumerate(zip(good_new, good_old)):
    a, b = new.ravel().astype(int)
    c, d = old.ravel().astype(int)
    mask = cv2.line(mask, (a, b), (c, d), color[i].tolist(), 2)
    frame = cv2.circle(frame, (a, b), 5, color[i].tolist(), -1)

img = cv2.add(frame, mask)

# Display the frames
cv2.imshow("Frame", img)
cv2.imshow("Foreground Mask", fgmask)
cv2.imshow("Gray_Frame", gray)
cv2.imshow("Threshold Frame", thresh_frame)
cv2.imshow("Colored_Frame", frame)

key = cv2.waitKey(1)
# Press q to stop the process
if key == ord('q'):
    if motion == 1:
        time.append(datetime.now())
    break
```

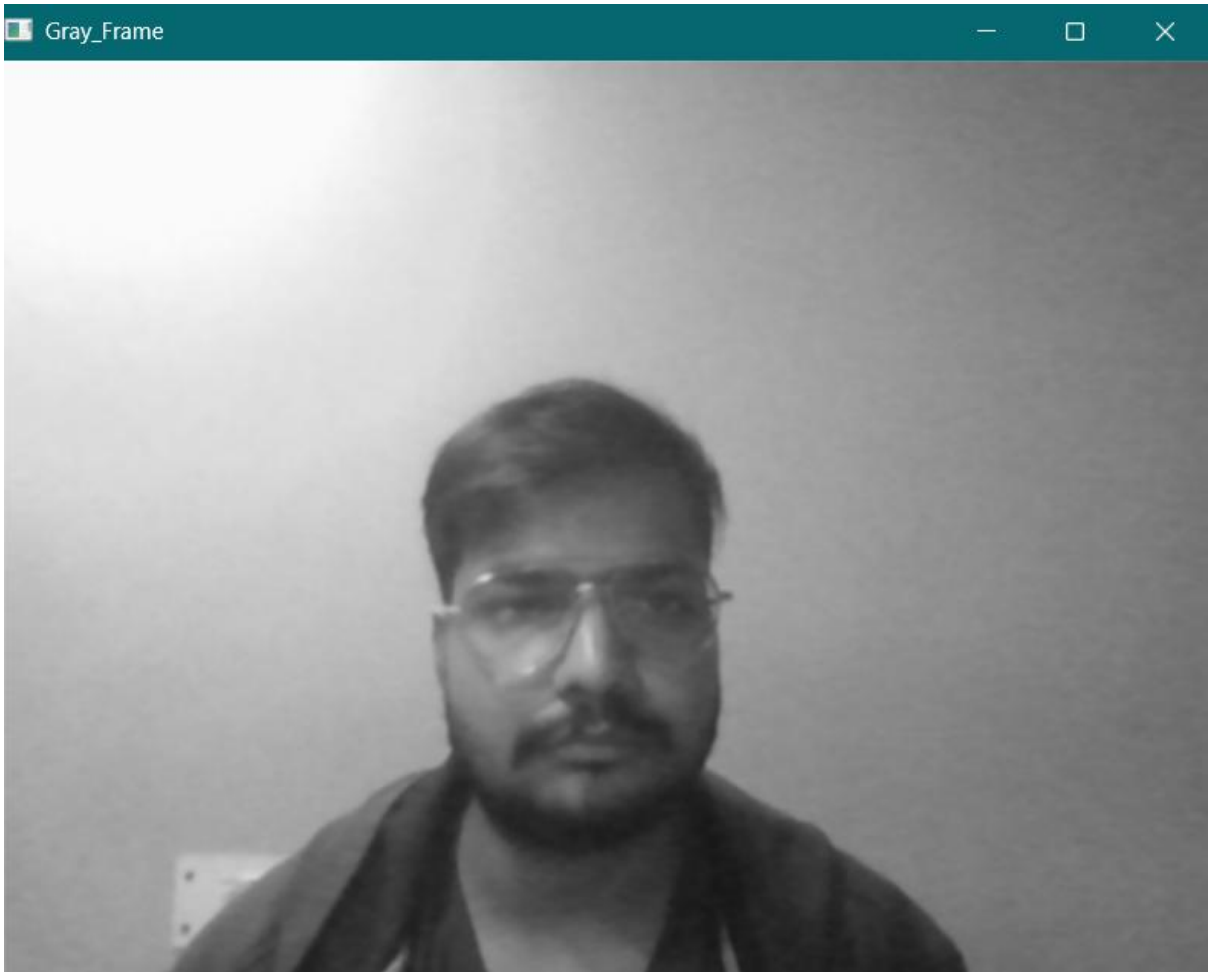
```
    # Now update the previous frame and previous points
    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1, 1, 2)

# Initialize an empty list to store the data
data = []

# Iterate through the time list in pairs
for i in range(0, len(time), 2):
    if pd.notna(time[i]) and pd.notna(time[i + 1]):
        # Append the pair to the data list
        data.append({"Start": time[i], "End": time[i + 1]})

# Convert the list to a DataFrame
df = pd.DataFrame(data)

# Print DataFrame to verify content
print(df)

# Creating a csv file in which time of movements will be saved
try:
    df.to_csv("FrameInMotion_time.csv")
    print("CSV file saved successfully.")
except Exception as e:
    print(f"Error saving CSV file: {e}")

video.release()

# close window
cv2.destroyAllWindows()
```

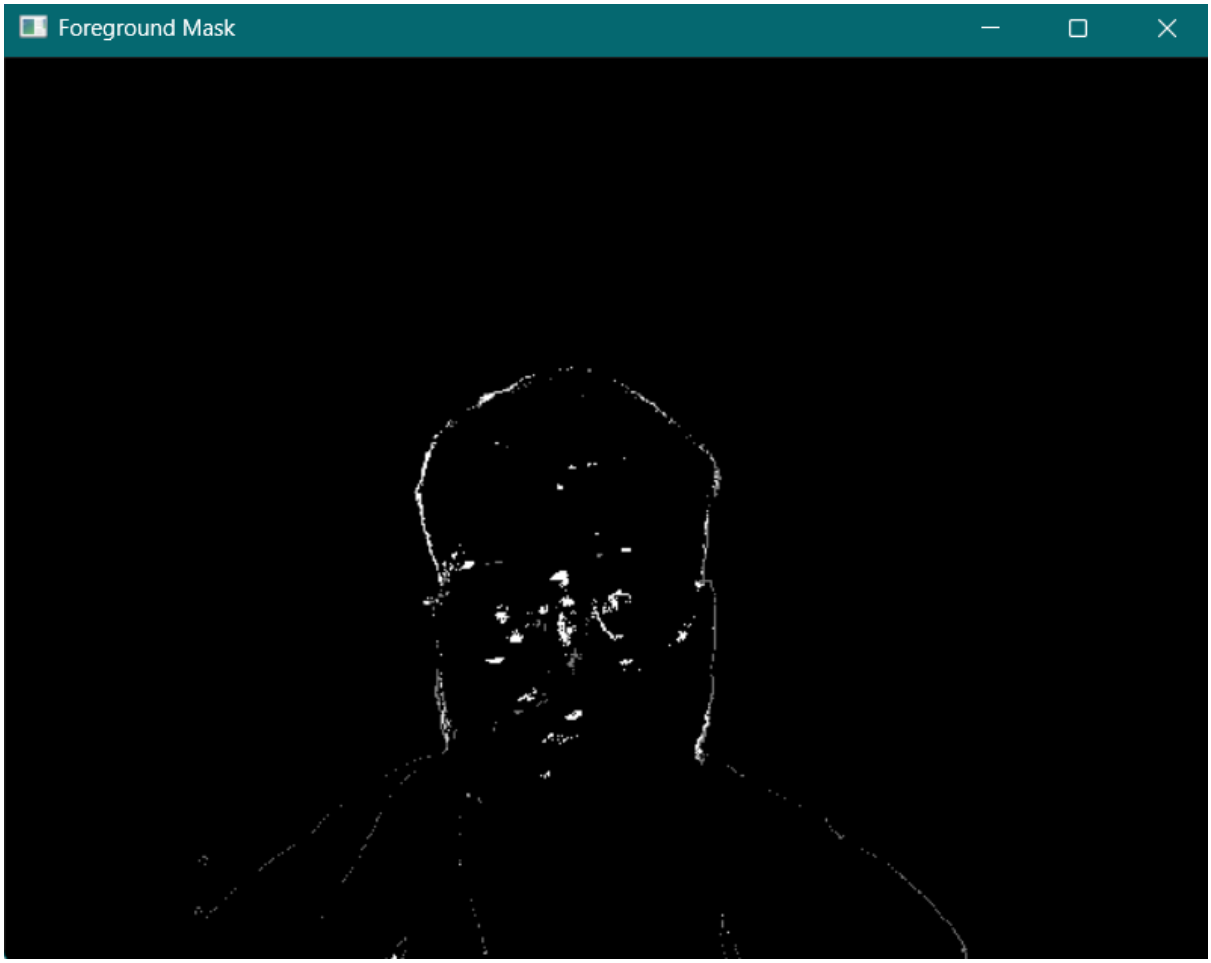
Output:

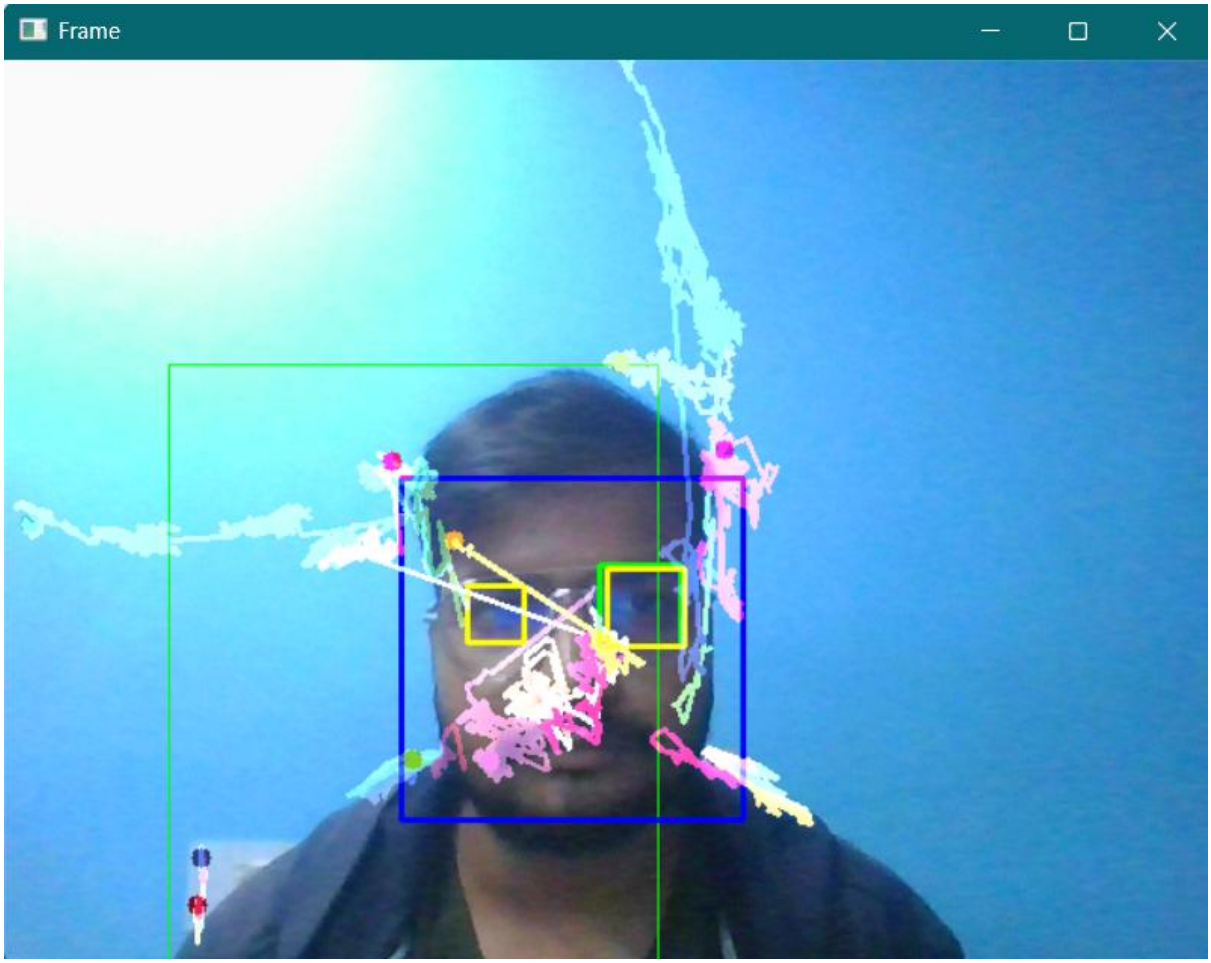
Video Type Output:




Threshold Frame


Gray_Frame


Foreground Mask


Frame

Program Output:

```
                       Start                       End
0 2024-11-16 14:15:13.476311 2024-11-16 14:15:13.711511
1 2024-11-16 14:15:13.855029 2024-11-16 14:15:14.089879
2 2024-11-16 14:15:14.152398 2024-11-16 14:15:14.248917
3 2024-11-16 14:15:14.313437 2024-11-16 14:15:14.556482
4 2024-11-16 14:15:14.635009 2024-11-16 14:15:14.795716
5 2024-11-16 14:15:16.949343 2024-11-16 14:15:16.982351
6 2024-11-16 14:15:24.138938 2024-11-16 14:15:46.915173
CSV file saved successfully.
```

Frame Motion CSV file output:

jupyter FrameInMotion_time.csv Last Checkpoint: 16 hours ago

File Edit View Settings Help

Delimiter: ,

		Start	End
1	0	2024-11-16 14:15:13.476311	2024-11-16 14:15:13.711511
2	1	2024-11-16 14:15:13.855029	2024-11-16 14:15:14.089879
3	2	2024-11-16 14:15:14.152398	2024-11-16 14:15:14.248917
4	3	2024-11-16 14:15:14.313437	2024-11-16 14:15:14.556482
5	4	2024-11-16 14:15:14.635009	2024-11-16 14:15:14.795716
6	5	2024-11-16 14:15:16.949343	2024-11-16 14:15:16.982351
7	6	2024-11-16 14:15:24.138938	2024-11-16 14:15:46.915173