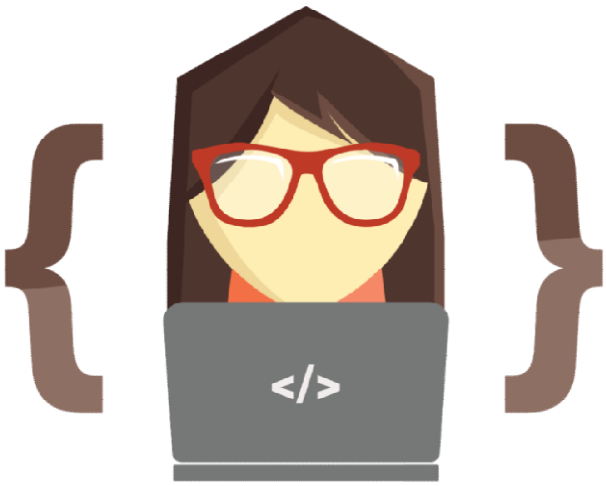


FACIAL RECOGNITION AND TRACKING

A DAY WISE PROJECT REPORT

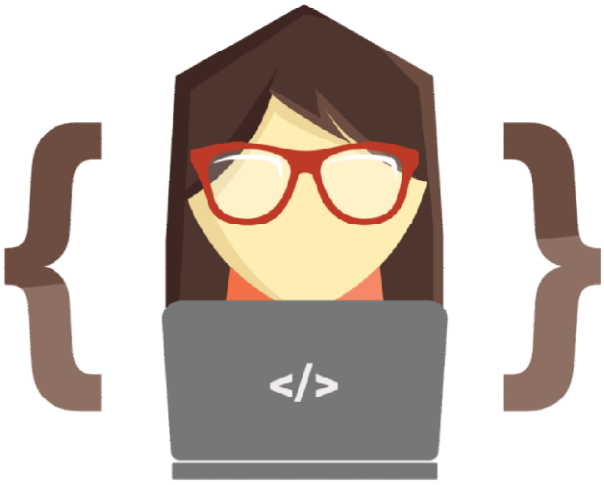
GIRL SCRIPT DEVELOPER TECH CAMP HACK-IN PROJECT

Hack –In is a week-long coding challenge in which the participants build a small-scale project using new technology.

DIBYENDU MAJI

Jalpaiguri Government Engineering College

ACKNOWLEDGEMENT



I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to our Project Mentor, Maitree Rawat, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project.

Furthermore I would also like to acknowledge with much appreciation the crucial role of the entire team of GirlScript Jaipur, for guiding the mentees in achieving the goal. I have to appreciate the guidance given by the GirlScript Jaipur Team in our project presentation that has improved our presentation skills thanks to their comment and advices.

ABSTRACT

Facial recognition is a biometric software application capable of uniquely identifying or verifying a person by comparing and analyzing patterns based on the person's facial contours. Facial recognition is mostly used for security purposes, though there is increasing interest in other areas of use. In fact, facial recognition technology has received significant attention as it has potential for a wide range of application related to law enforcement as well as other enterprises.

There are many advantages associated with facial recognition. Compared to other biometric techniques, facial recognition is of a non-contact nature. Face images can be captured from a distance and can be analyzed without ever requiring any interaction with the user/person. As a result, no user can successfully imitate another person. Facial recognition can serve as an excellent security measure for time tracking and attendance. Facial recognition is also cheap technology as there is less processing involved, like in other biometric techniques.

Facial Detection can be taken to next level by use of trackers using OpenCV as it is the fastest way to detect any object when multiple frames are captured. For every frame, detection of object will be a slow process whereas tracking it will be efficient.

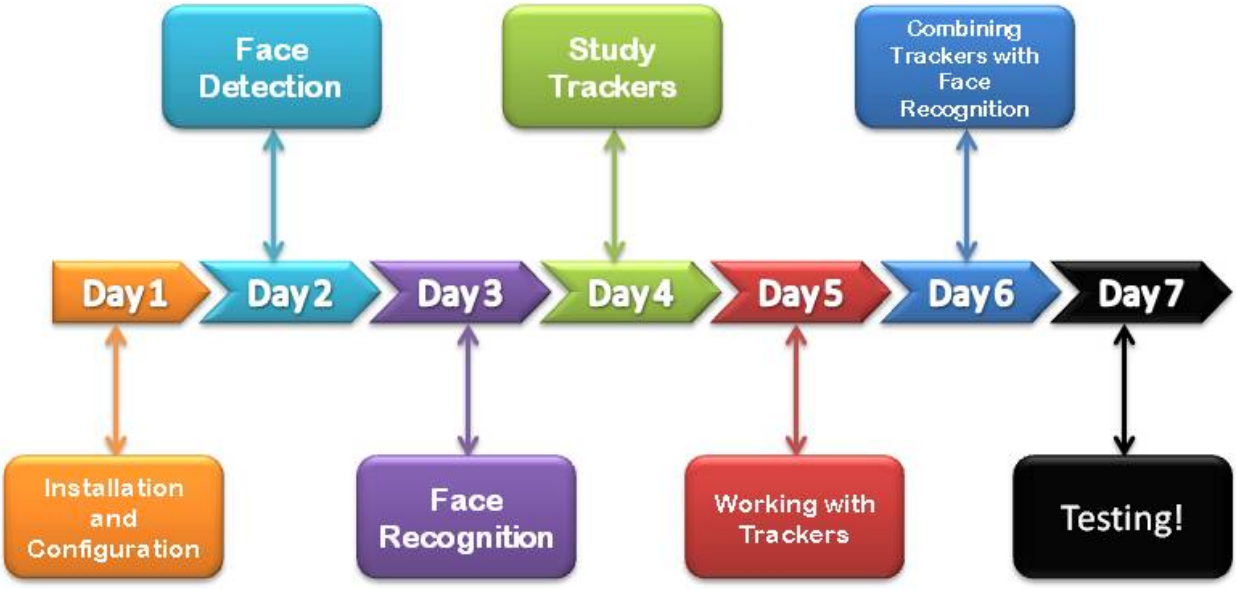
TECHNOLOGY STACK

1. SUBLIME TEXT(EDITOR)
2. PYTHON 3.7.2
3. OPENCV
4. PIP (PACKAGE MANAGEMENT SYSTEM)
5. IMUTILS VIDEO
6. MULTI-OBJECT TRACKER (KCF)
7. PIL (PYTHON IMAGING LIBRARY)

DAY 1: CREATING TIMELINE

- Objective: To decide a timeline about daily task to be performed each day and follow it accordingly.
- Installation: OpenCV is installed inside python and other necessary packages are installed.

- Timeline Image:



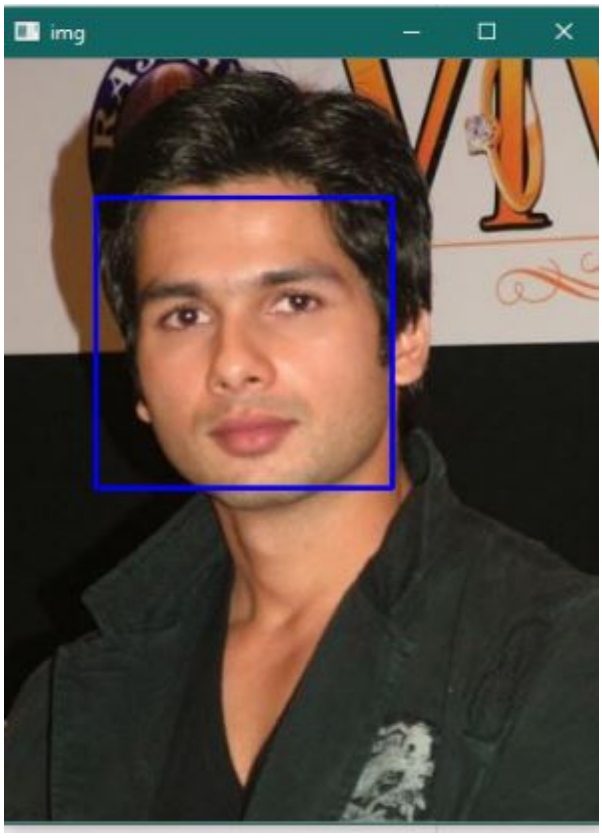
DAY 2: FACE DETECTION

- Objective: Start with Face Recognition using OpenCV in Python 3.5 and above and detect face Haar Cascade.

- Images:

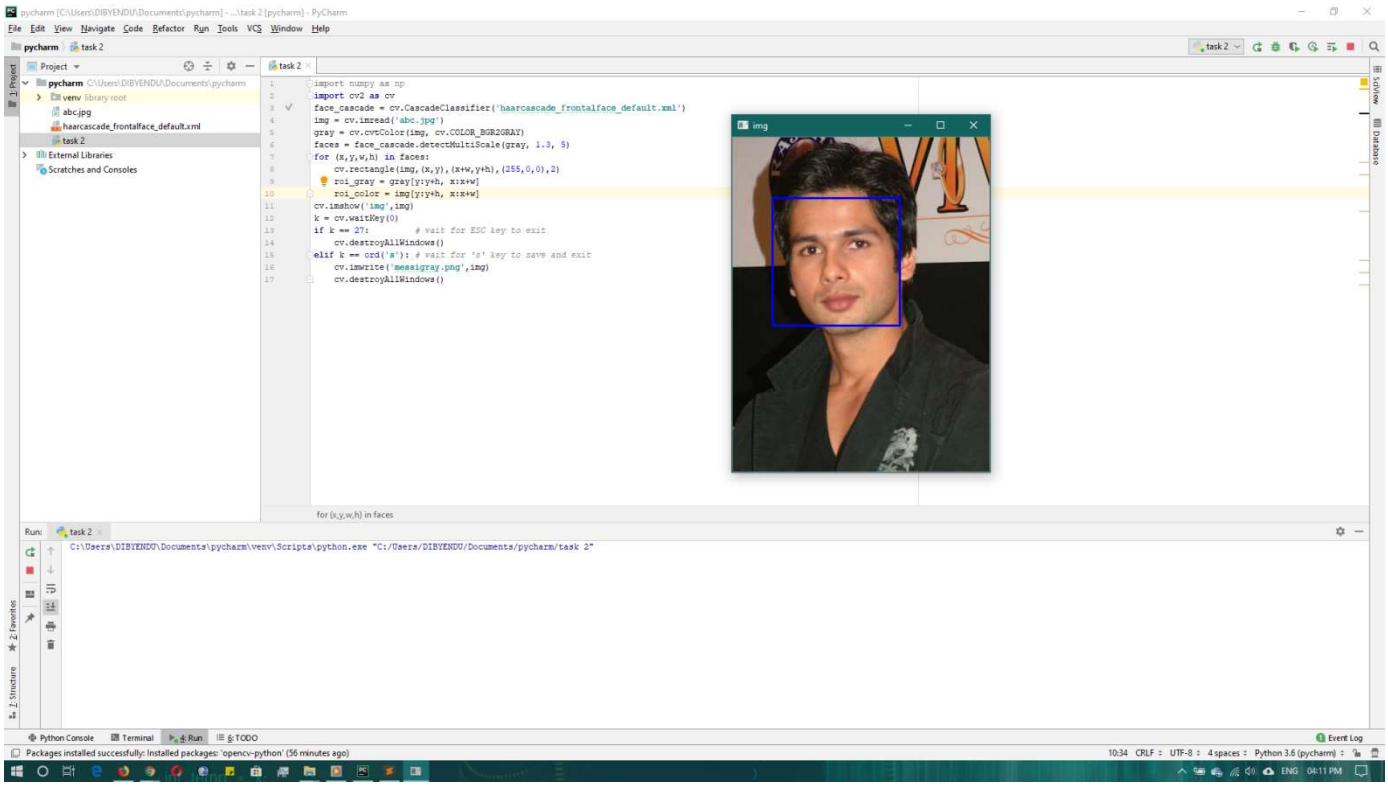
Original Image

Face Detected



❖ Code:

```
import numpy as np
import cv2 as cv
face_cascade = cv.CascadeClassifier('haarcascade_frontalface_default.xml')
img = cv.imread('abc.jpg')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    cv.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
cv.imshow('img',img)
k = cv.waitKey(0)
if k == 27:         # wait for ESC key to exit
    cv.destroyAllWindows()
elif k == ord('s'): # wait for 's' key to save and exit
    cv.imwrite('messigray.png',img)
    cv.destroyAllWindows()
```



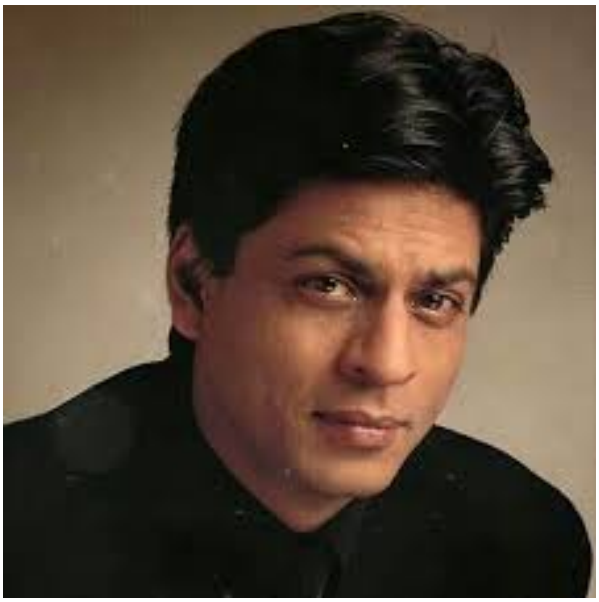
Sample Output Screenshots

DAY 3: FACE RECOGNITION

- Objective: To remember the face by training and storing it with some specific id, such that when similar faces are tested then the id can be displayed after recognition.

- Images:

Test Image 1



Test Image 2

CODE

- FACE RECOGNITION:

```
import cv2
import os
import numpy as np

def faceDetection(test_img):
    gray_img=cv2.cvtColor(test_img,cv2.COLOR_BGR2GRAY)
    face_haar_cascade=cv2.CascadeClassifier('HaarCascade/haarcascade_frontalface_default.xml')
    faces=face_haar_cascade.detectMultiScale(gray_img,scaleFactor=1.32,minNeighbors=5)

    return faces,gray_img

def labels_for_training_data(directory):
    faces=[]
    faceID=[]

    for path,subdirnames,filenames in os.walk(directory):
        for filename in filenames:
            if filename.startswith("."):
                print("Skipping system file")
                continue

            id=os.path.basename(path)
            img_path=os.path.join(path,filename)
            print("img_path:",img_path)
            print("id:",id)
            test_img=cv2.imread(img_path)
            if test_img is None:
                print("Image not loaded properly")
                continue
            faces_rect,gray_img=faceDetection(test_img)
            if len(faces_rect)!=1:
               continue
            (x,y,w,h)=faces_rect[0]
            roi_gray=gray_img[y:y+w,x:x+h]
            faces.append(roi_gray)
            faceID.append(int(id))
    return faces,faceID

def train_classifier(faces,faceID):
    face_recognizer=cv2.face.LBPHFaceRecognizer_create()
    face_recognizer.train(faces,np.array(faceID))
    return face_recognizer

#Below function draws bounding boxes around detected face in image
def draw_rect(test_img,face):
    (x,y,w,h)=face
    cv2.rectangle(test_img,(x,y),(x+w,y+h),(255,0,0),thickness=5)

#Below function writes name of person for detected label
def put_text(test_img,text,x,y):
    cv2.putText(test_img,text,(x,y),cv2.FONT_HERSHEY_DUPLEX,2,(255,0,0),4)
```

- RESIZING IMAGE

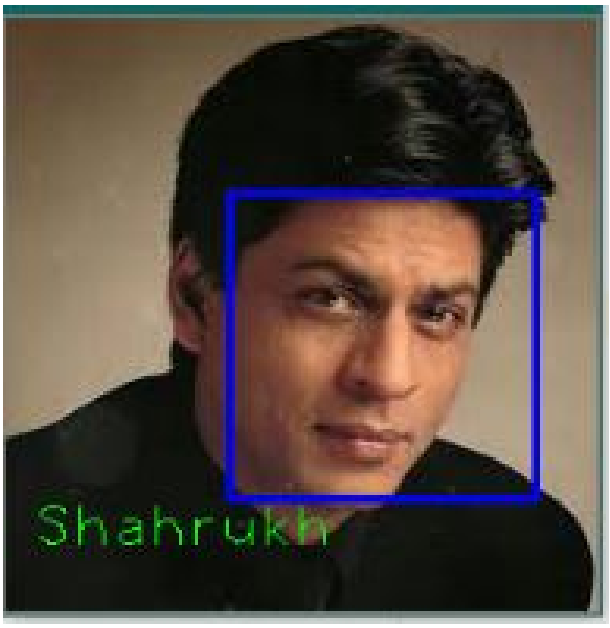
```
import cv2
import os
import numpy as np

count=0

for path, subdirnames, filenames in os.walk("trainingImages"):

    for filename in filenames:
      if filename.startswith("."):
        print("Skipping File:",filename)
        continue
      img_path=os.path.join(path, filename)
      print("img_path",img_path)
      id=os.path.basename(path)
      img = cv2.imread(img_path)
      if img is None:
        print("Image not loaded properly")
        continue
      resized_image = cv2.resize(img, (100, 100))
      new_path="resizedTrainingImages"+"/"+str(id)
      print("desired path is",os.path.join(new_path, "frame%d.jpg" % count))
      cv2.imwrite(os.path.join(new_path, "frame%d.jpg" % count),resized_image)
      count += 1
```

- TESTER

```
import cv2
import os
import numpy as np
import faceRecognition as fr

test_img=cv2.imread('TestImages/test.jpg')
faces_detected,gray_img=fr.faceDetection(test_img)
print("faces_detected:",faces_detected)

faces,faceID=fr.labels_for_training_data('trainingImages')
face_recognizer=fr.train_classifier(faces,faceID)
face_recognizer.write('trainingData.yml')

name={0:"Shahrukh",1:"Salman"}

for face in faces_detected:
    (x,y,w,h)=face
    roi_gray=gray_img[y:y+h,x:x+h]
    label,confidence=face_recognizer.predict(roi_gray)
    print("confidence:",confidence)
    print("label:",label)
    fr.draw_rect(test_img,face)
    predicted_name=name[label]
    if(confidence>37):
        continue
    fr.put_text(test_img,predicted_name,x,y)

resized_img=cv2.resize(test_img,(1000,1000))
cv2.imshow("face dtecetion tutorial",resized_img)
cv2.waitKey(0)
cv2.destroyAllWindows
```

OUTPUT IMAGES:

Test Image 1 Detected



Test Image 2 Detected

DAY 4: STUDY OF TRACKERS

- Objective: To study the use of Trackers and types of trackers.

- Types: OpenCV includes eight separate object tracking implementations that you can use in your own computer vision applications.

They are:

1. **BOOSTING Tracker:** Based on the same algorithm used to power the machine learning behind Haar cascades (AdaBoost), but like Haar cascades, is over a decade old. This tracker is slow and doesn't work very well. Interesting only for legacy reasons and comparing other algorithms.
2. **MIL Tracker:** Better accuracy than BOOSTING tracker but does a poor job of reporting failure.
3. **KCF Tracker:** Kernelized Correlation Filters. Faster than BOOSTING and MIL. Similar to MIL and KCF, does not handle full occlusion well.
4. **CSRT Tracker:** Discriminative Correlation Filter (with Channel and Spatial Reliability). Tends to be more accurate than KCF but slightly slower.
5. **MedianFlow Tracker:** Does a nice job reporting failures; however, if there is too large of a jump in motion, such as fast moving objects, or objects that change quickly in their appearance, the model will fail.
6. **TLD Tracker:** I'm not sure if there is a problem with the OpenCV implementation of the TLD tracker or the actual algorithm itself, but the TLD tracker was incredibly prone to false-positives. I do not recommend using this OpenCV object tracker.
7. **MOSSE Tracker:** Very, *very* fast. Not as accurate as CSRT or KCF but a good choice if you need pure speed.

8. **GOTURN Tracker:** The only deep learning-based object detector included in OpenCV. It requires additional model files to run (will not be covered in this post). My initial experiments showed it was a bit of a pain to use even though it reportedly handles viewing changes well (my initial experiments didn't confirm this though). I'll try to cover it in a future post, but in the meantime,

- Uses: Usually tracking algorithms are faster than detection algorithms. The reason is simple. When you are tracking an object that was detected in the previous frame, you know a lot about the appearance of the object. You also know the location in the previous frame and the direction and speed of its motion. So in the next frame, you can use all this information to predict the location of the object in the next frame and do a small search around the expected location of the object to accurately locate the object. A good tracking algorithm will use all information it has about the object up to that point while a detection algorithm always starts from scratch. Therefore, while designing an efficient system usually an object detection is run on every n^{th} frame while the tracking algorithm is employed in the n-1 frames in between. Why don't we simply detect the object in the first frame and track subsequently? It is true that tracking benefits from the extra information it has, but you can also lose track of an object when they go behind an obstacle for an extended period of time or if they move so fast that the tracking algorithm cannot catch up. It is also common for tracking algorithms to accumulate errors and the bounding box tracking the object slowly drifts away from the object it is tracking. To fix these problems with tracking algorithms, a detection algorithm is run every so often. Detection algorithms are trained on a large number of examples of the object.

DAY 5: WORKING WITH TRACKERS

- Objective: To detect object in real time using trackers in OpenCV.

- Images:

`

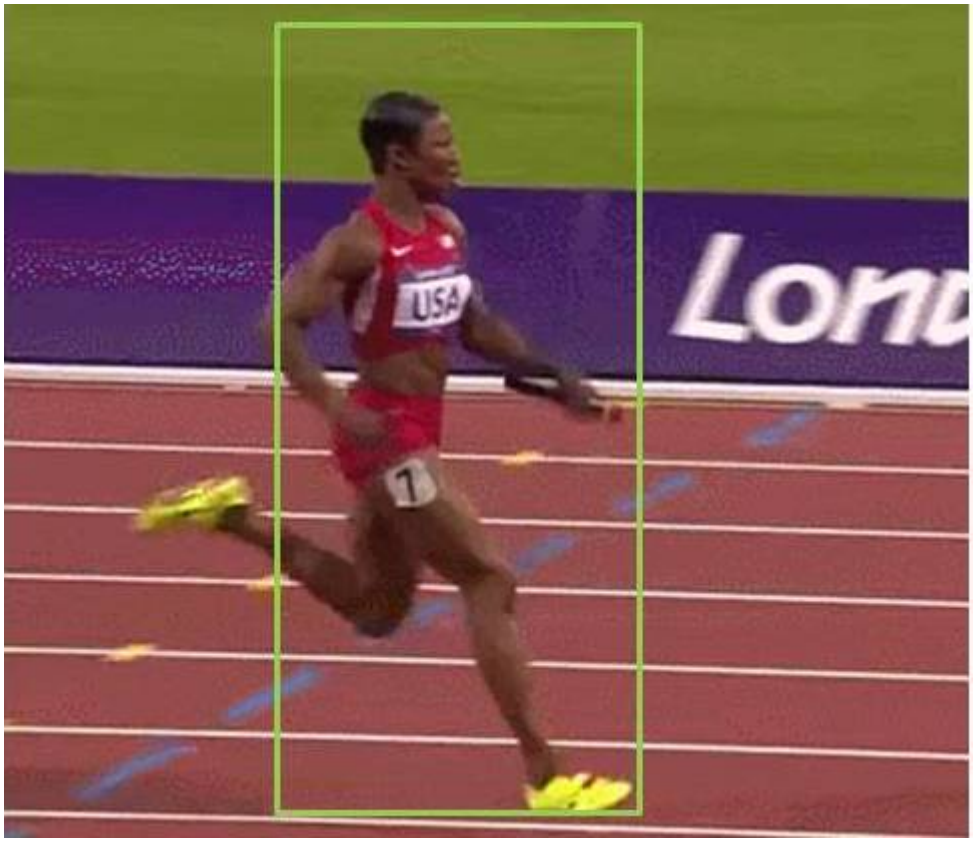
Input Video

❖ Code:

```
import sys
import cv2

if __name__ == '__main__':
    tracker_types = ['BOOSTING', 'MIL', 'KCF', 'TLD', 'MEDIANFLOW', 'GOTURN', 'CSRT']
    tracker_type = tracker_types[2]

    if tracker_type == 'BOOSTING':
        tracker = cv2.TrackerBoosting_create()
    if tracker_type == 'MIL':
        tracker = cv2.TrackerMIL_create()
    if tracker_type == 'KCF':
        tracker = cv2.TrackerKCF_create()
    if tracker_type == 'TLD':
        tracker = cv2.TrackerTLD_create()
    if tracker_type == 'MEDIANFLOW':
        tracker = cv2.TrackerMedianFlow_create()
    if tracker_type == 'GOTURN':
        tracker = cv2.TrackerGOTURN_create()
    if tracker_type == "CSRT":
        tracker = cv2.TrackerCSRT_create()

    # Read video
    video = cv2.VideoCapture("abcd.mp4")

    # Exit if video not opened.
    if not video.isOpened():
        print("Could not open video")
        sys.exit()

    # Read first frame.
    ok, frame = video.read()
    if not ok:
        print('Cannot read video file')
        sys.exit()

    # Define an initial bounding box
    bbox = (287, 23, 86, 320)

    # Uncomment the line below to select a different bounding box
    bbox = cv2.selectROI(frame, False)

    # Initialize tracker with first frame and bounding box
    ok = tracker.init(frame, bbox)
```

```
while True:
    # Read a new frame
    ok, frame = video.read()
    if not ok:
        break

    # Start timer
    timer = cv2.getTickCount()

    # Update tracker
    ok, bbox = tracker.update(frame)

    # Calculate Frames per second (FPS)
    fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer);

    # Draw bounding box
    if ok:
        # Tracking success
        p1 = (int(bbox[0]), int(bbox[1]))
        p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
        cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)
    else:
        # Tracking failure
        cv2.putText(frame, "Tracking failure detected", (100, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)

    # Display tracker type on frame
    cv2.putText(frame, tracker_type + " Tracker", (100, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

    # Display FPS on frame
    cv2.putText(frame, "FPS : " + str(int(fps)), (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

    # Display result
    cv2.imshow("Tracking", frame)

    # Exit if ESC pressed
    k = cv2.waitKey(1) & 0xff
    if k == 27: break
```

❖ Sample Output Screenshots:

❖ 

DAY 6: FINAL BUILD [COMBINING TRACKERS WITH FACE RECOGNITION]

- Objective: Finalized Improved Module for Face Recognition with OpenCV Trackers

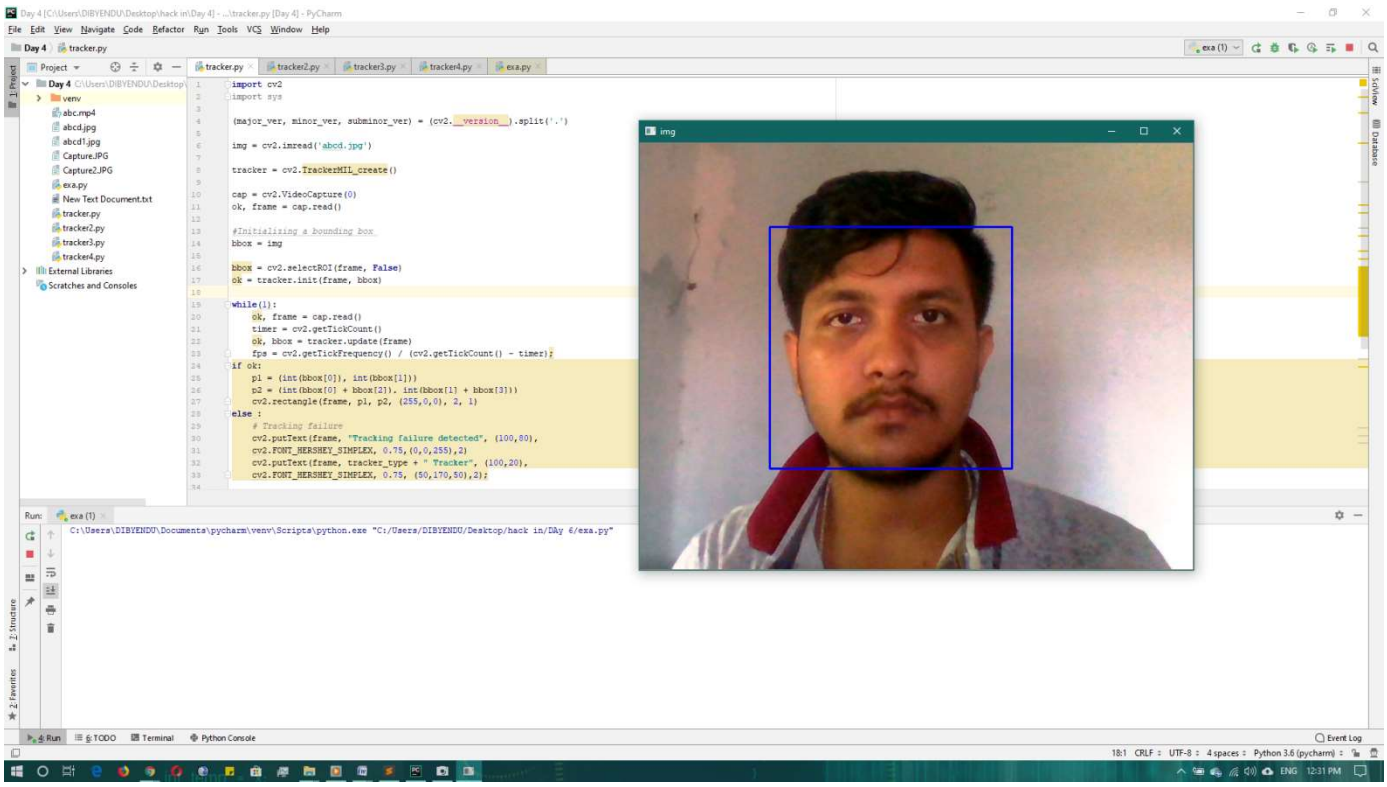
- Webcam View:

❖ Code:

❖

```
import cv2
import sys

(major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')

img = cv2.imread('abcd.jpg')

tracker = cv2.TrackerMIL_create()

cap = cv2.VideoCapture(0)
ok, frame = cap.read()

bbox = img

bbox = cv2.selectROI(frame, False)
ok = tracker.init(frame, bbox)

while(1):
    ok, frame = cap.read()
    timer = cv2.getTickCount()
    ok, bbox = tracker.update(frame)
    fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer);
if ok:
    p1 = (int(bbox[0]), int(bbox[1]))
    p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
    cv2.rectangle(frame, p1, p2, (255,0,0), 2, 1)
else :

    cv2.putText(frame, "Tracking failure detected", (100,80),
    cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2)
    cv2.putText(frame, tracker_type + " Tracker", (100,20),
    cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50),2);


cv2.putText(frame, "FPS : " + str(int(fps)), (100,50),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50), 2);

# Display result
cv2.imshow("Tracking", frame)

K = cv2.waitKey(1)
if k == 27:
    cv.destroyAllWindows()
elif k == ord('s'):
    cv2.imwrite('messigray.png',img)
    cv2.destroyAllWindows()
```

❖ Sample Output Screenshots:



FUTURE SCOPE

- It can be used for security and access such as facial biometrics
- Can be use to reduce thefts and robbery.
- Can be used to detect over speeding vehicles in smart traffic management system
- Can be used for detecting different objects and using with Machine Learning Algorithm
- It can be used for real time detection.

REFERENCES

1. https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/
2. https://realpython.com/face-detection-in-python-using-a-webcam/
3. https://www.datacamp.com/community/tutorials/face-detection-python-opencv
4. https://www.pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/
5. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html
6. https://pythonprogramming.net/haar-cascade-face-eye-detection-python-opencv-tutorial/
7. https://ieeexplore.ieee.org/abstract/document/8265863
8. https://www.learnopencv.com/multitracker-multiple-object-tracking-using-opencv-c-python/
9. https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/