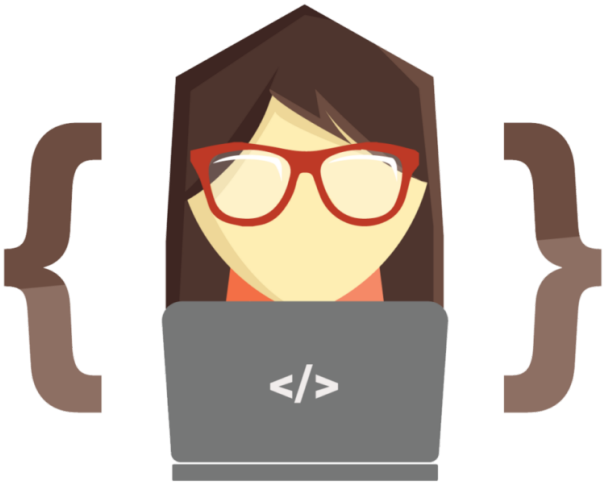


FACIAL RECOGNITION AND TRACKING

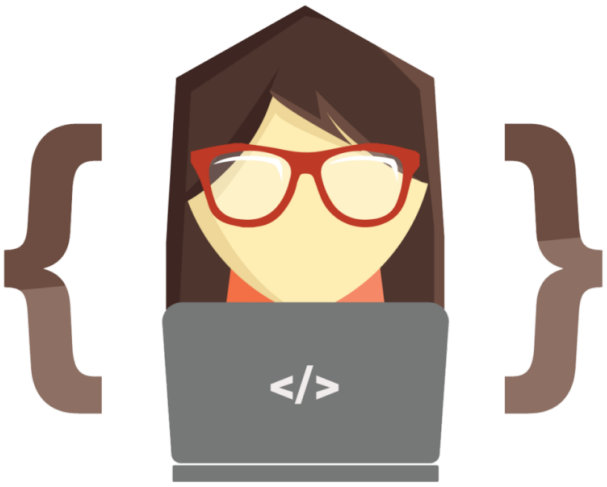
A DAY WISE PROJECT REPORT

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.

PRABHPREET SINGH
Guru Gobind Singh Indraprastha University

ACKNOWLEDGEMENT



I would like to express my special thanks of gratitude to my mentor, **Maitree Rawat** as well as to GirlScript Co-founder Anubha Maneshwar who gave me a golden opportunity to do this wonderful project on the topic **Face Recognition**, which also helped me in doing a lot of Research and I came to know about so many new things I am really thankful to them.

Also, I would also like to thank my friends, my fellow mentees and my parents who helped me a lot in finalising this project within the limited time frame.

ABSTRACT

A facial recognition system is a technology capable of identifying or verifying a person from a digital image or a video frame from a video source. There are multiple methods in which facial recognition systems work, but in general, they work by comparing selected facial features from given image with faces within a database. It is also described as a Biometric Artificial Intelligence based application that can uniquely identify a person by analysing patterns based on the person's facial textures and shapes.

The process of face recognition is performed in two steps. The first involves feature extraction and selection and, the second is the classification of objects.

Application

- Mobile platforms
- Social media
- Face ID
- ID Verification Solutions
- Deployment in security services
- Policing
- National security

TECHNOLOGY STACK

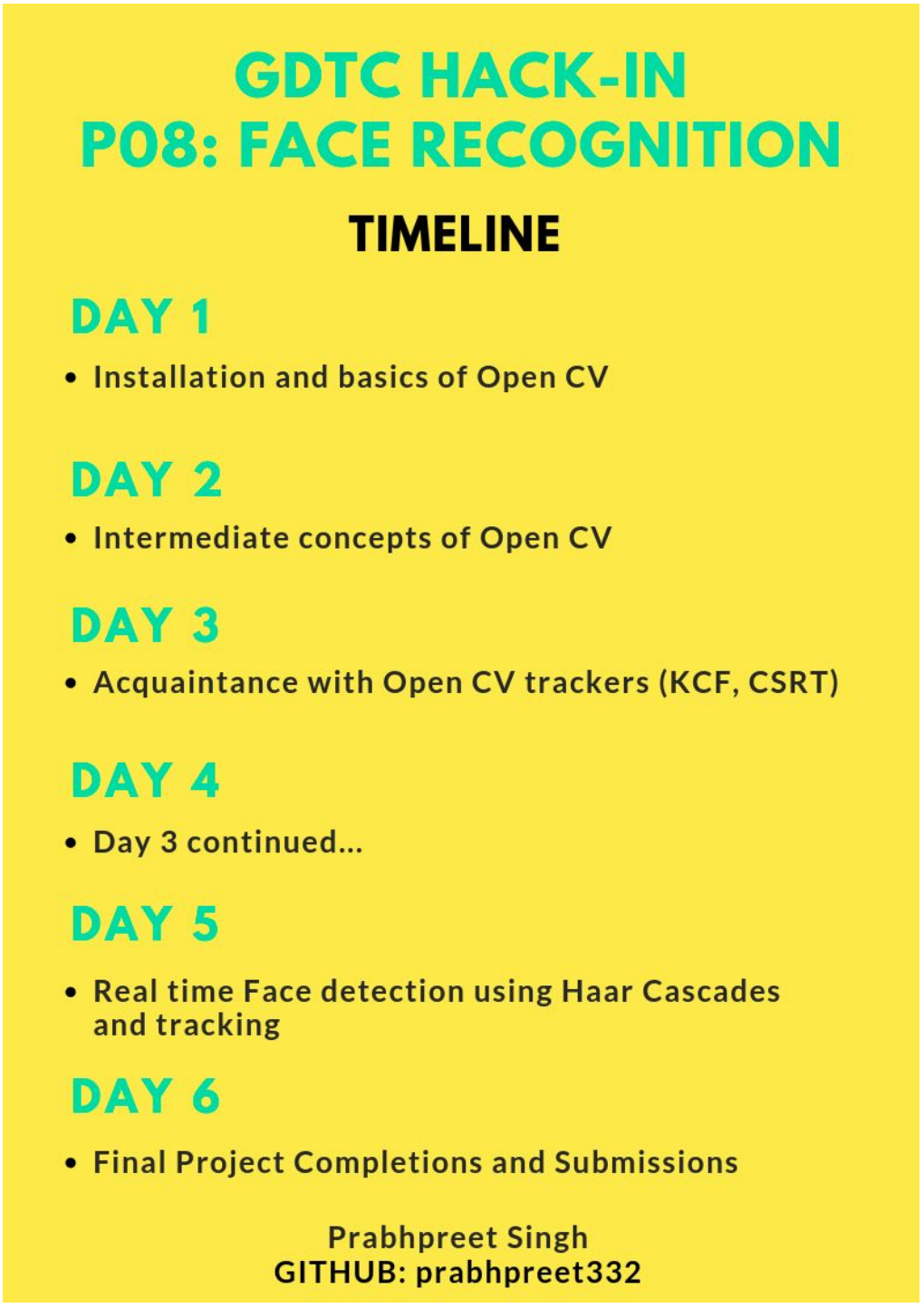
The following tools/technologies (and their versions) were used to complete the project :

- Jupyter Notebooks (5.2)
- Python (3.7)
- OpenCV (4.1)
- Numpy (1.16)
- Anaconda Package Manager (4.6)
- Pillow (6.6)
- Matplotlib (3.1)

DAY 1: CREATING TIMELINE

Objective: To create a roadmap for upcoming weeks progress

Timeline Image:



DAY 2: OPENCV CONCEPTS

Objective: Go through Intermediate concepts of OpenCV

Open CV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

Click a picture with OpenCV

In [1]:
```
import cv2
import numpy as np
import time
import matplotlib.pyplot as plt
```

Click a pic

In [11]:
```
capture = cv2.VideoCapture(0)
while True:
    vid = capture.read()
    im_path = 'captured_images/img.jpg'
    cv2.imwrite(im_path, vid[1])
    print("Image Clicked!")
    cv2.waitKey(1)
    break
capture.release()
cv2.destroyAllWindows()
```

```
Image Clicked!
```

Continuously click pictures with OpenCV

In [2]:

```
cap = cv2.VideoCapture(0)
num = 5 # This is the number of Images to be clicked

while True:
    ret,frame = cap.read()

    for count in range(num):
        im_path = f'captured_images/img_cont_{count}.jpg'
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        cv2.imwrite(im_path, frame)
        time.sleep(1)
        print(f"Image {count+1} captured")
        if count == num-1:
            break

    cv2.waitKey(1)
    break

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

```
Image 1 captured
Image 2 captured
Image 3 captured
Image 4 captured
Image 5 captured
```

Crop Image

In [6]:

```
import os
im_path = os.getcwd() + '/captured_images/test.jpg'
image = cv2.imread(im_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
print(np.shape(image))
row = np.shape(image)[0]
col = np.shape(image)[1]
plt.imshow(image)
plt.show()

```

```
(480, 640, 3)
```

DAY 3: OpenCV Trackers

Objective: Get Acquainted with OpenCV trackers

OpenCV includes eight separate object tracking implementations that we can use in our own computer vision applications.

I've included a brief highlight of each object tracker below:

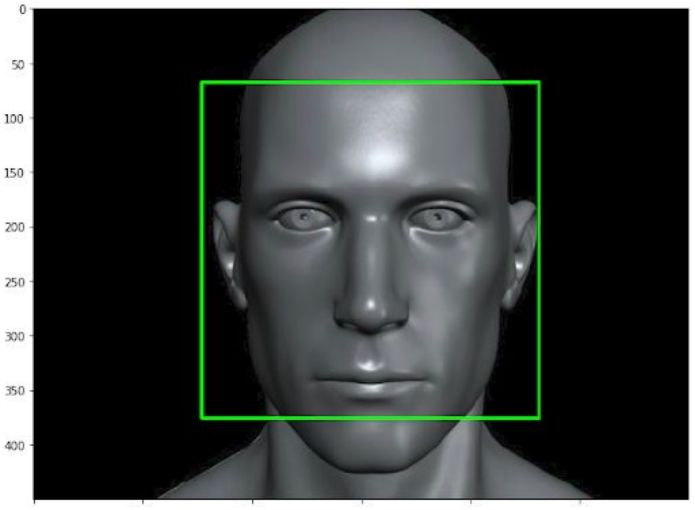
1. **BOOSTING Tracker**: Based on the same algorithm used to power the machine learning behind Haar cascades (AdaBoost). This tracker is slow and doesn't work very well.
2. **MIL Tracker**: Better accuracy than BOOSTING tracker but does a poor job of reporting failure.
3. **KCF Tracker**: Kernelized 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**: TLD tracker is incredibly prone to false-positives.
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.

We will be using KCF tracker and CSRT tracker for face detection and tracking in this project.

Face Detection using Haar Cascades

In [2]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

In [4]:
```
plt.figure(figsize=(18,8))
# load the cascade classifiers xml file
cascade_path = 'haarcascades/haarcascade_frontalface_default.xml'
face_cascade = cv2.CascadeClassifier(cascade_path)

# get image path
im_path = 'sample_images/'
images = ['human1.jpg', 'human2.png', 'human3.jpg']
im_path += images[2]

orig_img = cv2.imread(im_path)
img = orig_img
# covert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
# draw rectangel
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

plt.imshow(img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



DAY 4: CONTINUING WITH TRACKERS

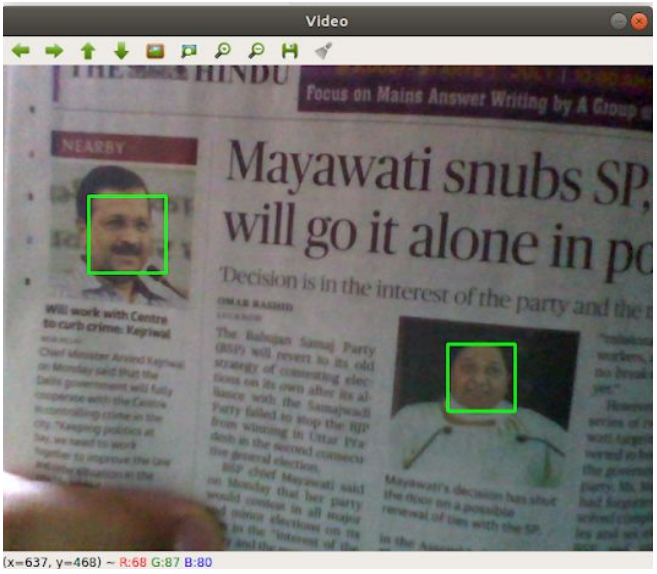
Objective: Implementing and using OpenCV trackers for Face-Recognition

Face Tracking using Haar Cascades

In [1]:
```
import numpy as np
import cv2
```

In [2]:
```
cascade_path = 'haarcascades/haarcascade_frontalface_default.xml'
face_cascade = cv2.CascadeClassifier(cascade_path)

video_capture = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

# draw a rectangle
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    cv2.imshow('Video', frame)

    if cv2.waitKey(1) == ord('q'):
        break

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

Output



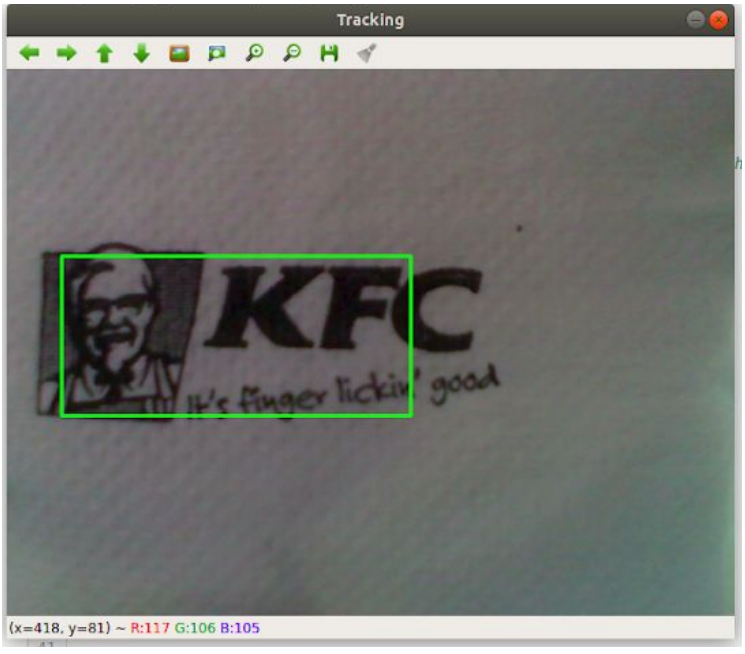
Tracking using CSRT tracker

In [8]:

```
tracker = cv2.TrackerCSRT_create()

video = cv2.VideoCapture(0)

ret, frame = video.read()

frame_width = np.shape(frame)[1]
frame_height = np.shape(frame)[0]

if not ret:
    print( 'Cannot read video file')
    sys.exit(1)

# bounding box
x = 1
# tracking_dimensions = (frame_height//2 - x,frame_width//2 -x,frame_height//2 + x, frame_width//2 + x)
# bbox = tracking_dimensions#(287, 23, 86, 320)
bbox = cv2.selectROI(frame, False)

# initialising tracker with frame and bounding box
oretk = tracker.init(frame, bbox)

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


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


    # draw bounding box
    if ret:
        p1 = (int(bbox[0]), int(bbox[1]))
        p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
        cv2.rectangle(frame, p1, p2, (0,255,0), 2, 1)



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

    # Exit if ESC pressed
    k = cv2.waitKey(1) & 0xff
    if k == 27 :
        print("esc key prssed")
        break



video.release()
cv2.destroyAllWindows()
```

```
esc key prssed
```

Output



DAY 5: FACE RECOGNITION AND TRACKING

Objective: Real Time Face Detection and Tracking

Tracking using KCF tracker

In [9]:

```
tracker = cv2.TrackerKCF_create()

video = cv2.VideoCapture(0)

ret, frame = video.read()

frame_width = np.shape(frame)[1]
frame_height = np.shape(frame)[0]

if not ret:
    print( 'Cannot read video file')
    sys.exit(1)

# bounding box
x = 1
# tracking_dimensions = (frame_height//2 - x,frame_width//2 -x,frame_height//2 + x, frame_width//2 + x)
# bbox = tracking_dimensions#(287, 23, 86, 320)
bbox = cv2.selectROI(frame, False)

# initialising tracker with frame and bounding box
oretk = tracker.init(frame, bbox)

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


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


    # draw bounding box
    if ret:
        p1 = (int(bbox[0]), int(bbox[1]))
        p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
        cv2.rectangle(frame, p1, p2, (0,255,0), 2, 1)



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

    # Exit if ESC pressed
    k = cv2.waitKey(1) & 0xff
    if k == 27 :
        print("esc key prssed")
        break


video.release()
cv2.destroyAllWindows()
```

```
esc key prssed
```

Output

FUTURE SCOPE

- **Can be used with Raspberry Pi with webcam to detect robbery.**
- **Can be used with Quadrone to track real time person.**
- **Can be used to detect over speeding vehicles in smart traffic management system**

REFERENCES

To complete this project following references were used :

- https://docs.opencv.org/4.1.0/index.html
- https://github.com/girlscriptjaipur/GDTC_Hack-In/tree/master/P08_FaceRecognition/DAY%201
- https://github.com/girlscriptjaipur/GDTC_Hack-In/tree/master/P08_FaceRecognition/DAY%202
- https://github.com/girlscriptjaipur/GDTC_Hack-In/tree/master/P08_FaceRecognition/Day%203
- https://github.com/girlscriptjaipur/GDTC_Hack-In/blob/master/P08_FaceRecognition/RESOURCES/resources
- https://www.pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/