Vidyasagar University

An approach to faceswap of images using Opencv python

By

NAME: **ADITYA DAS**

ROLL_NO.: **VU/PG/20/32/02-IVS-0026**

REGISTRATION NO.: **01079 of 2020-21**

NAME: **SUPORNA SARKAR**

ROLL_NO.: **VU/PG/20/32/02-IVS-0017**

REGISTRATION NO.: **01070**

NAME: **ANANYA GHOSH**

ROLL_NO.: **VU/PG/20/32/02-IVS-0035**

REGISTRATION NO.: **0000105**

NAME: **MOUMITA MAITY**

ROLL_NO.: **VU/PG/20/32/02-IVS-0053**

REGISTRATION NO.: **0000543**

Under the supervised mentor

Dr. Partho Choudhury

ABSTRACT

In case of image processing the package called OpenCV or OpenCV3 is the
efficient way to edit image or video. It takes input and process those input depends upon their pixel & density and applying algorithm, it generate outputs. Experiments with this tool were performed using two person's face dlib 68' point;

A comparative analytical approach was done to determine how the ensemble technique can be applied for improving output in case of swap two faces. The focus of this paper is not only on increasing the accuracy of the output of weak algorithms, but also on the implementation of the proper algorithm with a particular input. The results of the study indicate that ensemble technique, such as dlib 68' point method, facial land marking, triangle wrapping these are effectively improving the accuracy of the output and exhibit satisfactory performance in case of image processing. In a case study a maximum of 7% accuracy for the weak algorithm to use such another method like in our case dlib 68' point, facial land marking, triangle wrapping etc.

The performance of the process was future enhance with a feature selection implementation and the results showed significant improvement in prediction accuracy.

Chapter-1

1.1 PURPOSE OF AI IN FACE-Swapping:

When faceswapping was first developed and published, the technology was groundbreaking, it was a huge step in AI development. It was also completely ignored outside of academia because the code was confusing and fragmentary. It required a thorough understanding of complicated AI techniques and took a lot of effort to figure it out. Until one individual brought it together into a single, cohesive collection. It ran, it worked, and as is so often the way with new technology emerging on the internet, it was immediately used to create inappropriate content. Despite the inappropriate uses the software was given originally, it was the first AI code that anyone could download, run and learn by experimentation without having a Ph.D. in math, computer theory, psychology, and more. Before "deepfakes" these techniques were like black magic, only practiced by those who could understand all of the inner workings as described in esoteric and endlessly complicated books and papers. "Deepfakes" changed all that and anyone could participate in AI development. To us, developers, the release of this code opened up a fantastic learning opportunity. It allowed us to build on ideas developed by others, collaborate with a variety of skilled coders, experiment with AI whilst learning new skills and ultimately contribute towards an emerging technology which will only see more mainstream use as it progresses. Are there some out there doing horrible things with similar software? Yes. And because of this, the developers have been following strict ethical standards. Many of us don't even use it to create videos, we just tinker with the code to see what it does. Sadly, the media concentrates only on the unethical uses of this software. That is, unfortunately, the nature of how it was first exposed to the public, but it is not representative of why it was created, how we use it now, or what we see in its future. Like any technology, it can be used for good or it can be abused. It is our intention to develop FaceSwap in a way that its potential for abuse is minimized whilst maximizing its potential as a tool for learning, experimenting and, yes, for legitimate faceswapping. We are not trying to denigrate celebrities or to demean anyone. We are programmers, we are engineers, we are Hollywood VFX artists, we are activists, we are hobbyists, we are human beings. To this end, we feel that it's time to come out with a standard statement of what this software is and isn't as far as us developers are concerned. FaceSwap is not for creating inappropriate content. FaceSwap is not for changing faces without consent or with the intent of hiding its use. FaceSwap is not for any illicit, unethical, or questionable purposes.FaceSwap exists to experiment and discover AI techniques, for social or political commentary, for movies, and for any number of ethical and reasonable uses. We are very troubled by the fact that FaceSwap can be used for unethical and disreputable things. However, we support the development of tools and techniques that can be used ethically as well as provide education and experience in AI for anyone who wants to learn it hands-on. We will take a zero

tolerance approach to anyone using this software for any unethical purposes and will actively discourage any such uses.

1.2 Python as a programming language

- Python In Artificial Intelligence

One of the key features of python language is its simplicity in code writing. It uses 1/5th of the code when compared to other object oriented programs. This factor makes it the most sort after language used in trending domains like AI. AI has a wide horizon under which it deals with machine learning and deep learning.

Python has a variety of libraries that appeal to the needs of any programmer. It has some prebuilt libraries such as Numpy, SciPy, Pybrain etc., which are for advance and scientific computing. Python is platform independent, which makes it quite flexible in interfacing between other technologies. In addition, the current user base of the language is very diverse. Most python developers share queries and solutions on portals, which make it a comprehensive knowledge resource as well.

The language not only applies OOPs concepts but incorporates a scripting approach as well. There are numerous IDEs (Integrated Development Environment) like PyCharm, which allows users to carry out complex codes and algorithms of AI related projects. In an AI's SDLC (Software Development Life Cycle) phase like testing, debugging and development, it becomes a cakewalk, when compared against other contemporary programming languages like Java, Javascript and Pearl.

These languages would definitely yield desired results but would make tasks cumbersome. Hence, looking at the numerous advantages of python, there is no doubt that it plays a crucial aspect in today's AI technologies

1.3 Python Packages:

Now that we know how important Python is to the world and us. Let us deep dive into learning some of the technical aspects of the programming language. The below illustrated topics are rudimentary and would be easy to grasp.

- Python use in machine learning

Machine Learning, as the name suggests, is the science of programming a computer by which they are able to learn from different kinds of data. A more general definition given by Arthur Samuel is – "Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed." They are typically used to solve various types of life problems.

In the older days, people used to perform Machine Learning tasks by manually coding all the algorithms and mathematical and statistical formula. This made the process time consuming, tedious and inefficient. But in the modern days, it is become very much easy and efficient compared to the olden days by various python libraries, frameworks, and modules. Today, Python is one of the most popular programming languages for this task and it has replaced many languages in the industry, one of the reason is its vast collection of libraries. Python libraries that used in Machine Learning are:

- Numpy
- Scipy
- OpenCV

- NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow uses NumPy internally for manipulation of Tensors.

```
# Python program using NumPy # for some basic mathematical # operations
import numpy as np
# Creating two arrays of rank 2 x = np.array([[1, 2], [3, 4]])
 y = np.array([[5, 6], [7, 8]])
# Creating two arrays of rank 1 v = np.array([9, 10])
w = np.array([11, 12])
# Inner product of vectors print(np.dot(v, w), "\n")
# Matrix and Vector product print(np.dot(x, v), "\n")
# Matrix and matrix product print(np.dot(x, y))
```

```
Output:
219

[29 67]

[[19 22]
 [43 50]]
```

Figure 1.1 :The output of above program

- SciPy is a very popular library among Machine Learning enthusiasts as it contains different modules for optimization, linear algebra, integration and statistics. There is a difference between the SciPy library and the SciPy stack. The SciPy is one of the core packages that make up the SciPy stack. SciPy is also very useful for image manipulation.

```
# Python script using Scipy # for image manipulation
from scipy.misc import imread, imsave, imresize
 # Read a JPEG image into a numpy array
img = imread('D:/Programs / cat.jpg') # path of the image print(img.dtype, img.shape)

# Tinting the image
```

```
img_tint = img * [1, 0.45, 0.3]

# Saving the tinted image
imsave('D:/Programs / cat_tinted.jpg', img_tint)

# Resizing the tinted image to be 300 x 300 pixels img_tint_resize = imresize(img_tint, (300, 300))

# Saving the resized tinted image
imsave('D:/Programs / cat_tinted_resized.jpg', img_tint_resize)
```

output:

Original image:

Tinted image:

Resized tinted image:

Figure 1.2 output of above program

- OpenCV is a Python library which is designed to solve computer vision problems. OpenCV was originally developed in 1999 by Intel but later it was supported by Willow Garage. OpenCV supports a wide variety of programming languages such as C++, Python, Java etc. Support for multiple platforms including Windows, Linux, and MacOS.OpenCV Python is nothing but a wrapper class for the original C++ library to be used with Python. Using this, all of the OpenCV array structures gets converted to/from NumPy arrays.This makes it easier to integrate it with other libraries which use NumPy. For example, libraries such as SciPy and Matplotlib.

```
Import cv2

# colored Image

Img = cv2.imread (&ldquo;Penguins.jpg&rdquo;,1)


# Black and White (gray scale)

Img_1 = cv2.imread (&ldquo;Penguins.jpg&rdquo;,0)
```

1.4 Packages use in project:

1.4.1 NUMPY :

NumPy – which stands for numerical Python; It is a library consisting of multidimensional, array objects and collection of routines on processing those always. NumPy helps to perform mathematical and logical operation on array.

Operation using NumPy:

- Mathematical and logical on array.
- Furrier transforms and routines for shape manipulation.
- Operation related to linear algebra and in build functions for linear Algebra and random number generation..NumPy is often used along with packages like spicy (scientific Python) and MatPlotlib (Plotting library) .Import NumPy as me the most import object define in NumPy is an n dimensional array type called an array.

import NumPy a np

a = np.array ([1, 2, 3]) Print a

Output: -

[1, 2, 3]

import NumPy as np

a= np.array ([[12, 3],[4,5,6]])

Print a.Shape

Output:

(2, 3)

Chapter-2

2.1 Tools & packages using in project

2.1.1 OpenCV

- **What Is Computer Vision?**

We all use Facebook, correct? Let us say you and your friends went on a vacation and you clicked a lot of pictures and you want to upload them on Facebook and you did. But now, wouldn't it take so much time just to find your friends faces and tag them in each and every picture? Well, Facebook is intelligent enough to actually tag people for you.

So, how do you think the auto tag feature works? In simple terms, it works on computer vision.

Computer Vision is an interdisciplinary field that deals with how computers can be made to gain a high-level understanding from digital images or videos.

The idea here is to automate tasks that the human visual systems can do. So, a computer should be able to recognize objects such as that of a face of a human being or a lamppost or even a statue.

- **How Does A Computer Read An Image?**

Consider the below image:

Fig:2.1

We can figure out that it is an image of the New York Skyline. But, can a computer find this out all on its own? The answer is no! The computer reads any image as a range of values

between 0 and 255. For any color image, there are 3 primary channels – Red, green and blue. How it works is pretty simple A matrix is formed for every primary color and later these matrices combine to provide a Pixel value for the individual R, G, B colors. Each element of the matrices provide data pertaining to the intensity of brightness of the pixel.

Consider the following image:

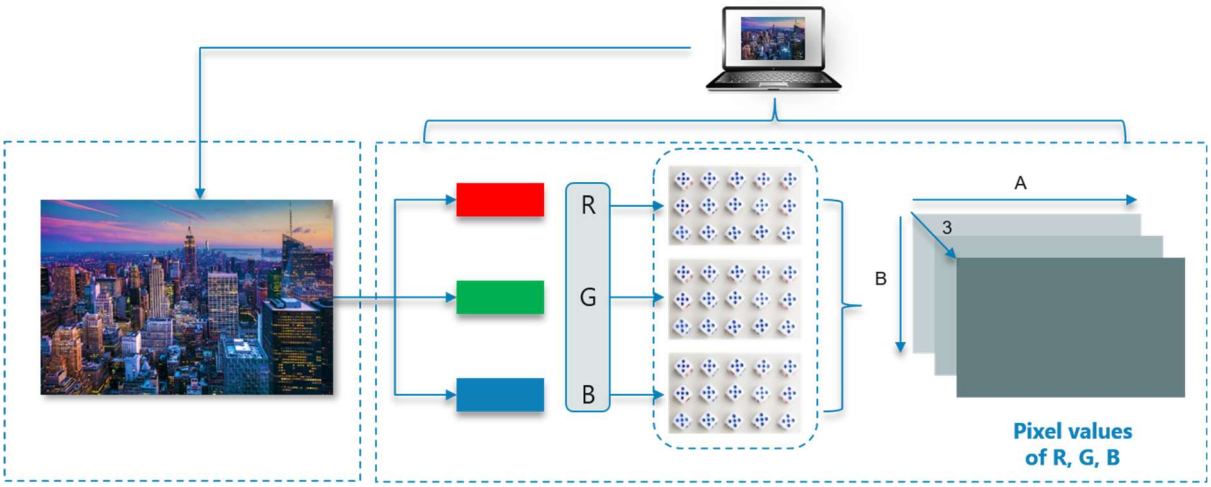


Fig.: 2.2

As shown, the size of the image here can be calculated as B x A x 3.

Note: For a black-white image, there is only one single channel.

Next up on this OpenCV Python Tutorial blog, let us look at what OpenCV actually is.

- **What Is OpenCV?**

OpenCV is a Python library which is designed to solve computer vision problems. OpenCV was originally developed in 1999 by Intel but later it was supported by Willow Garage.

OpenCV supports a wide variety of programming languages such as C++, Python, Java etc. Support for multiple platforms including Windows, Linux, and MacOS.

```
Import cv2
1
2
3
4# colored Image
5
6Img = cv2.imread (&ldquo;Penguins.jpg&rdquo;,1)
7
8
9
10# Black and White (gray scale)
11
12Img_1 = cv2.imread (&ldquo;Penguins.jpg&rdquo;,0)
13
```

OpenCV Python is nothing but a wrapper class for the original C++ library to be used with Python. Using this, all of the OpenCV array structures gets converted to/from NumPy arrays.

This makes it easier to integrate it with other libraries which use NumPy. For example, libraries such as SciPy and Matplotlib.

Next up on this OpenCV Python Tutorial blog, let us look at some of the basic operations that we can perform with OpenCV.

- ## Basic Operations With OpenCV?

Let us look at various concepts ranging from loading images to resizing them and so on.

- ## Loading an image using OpenCV:

As seen in the above piece of code, the first requirement is to import the OpenCV module.

Later we can read the image using **imread** module. The 1 in the parameters denotes that it is a color image. If the parameter was 0 instead of 1, it would mean that the image being imported is a black and white image. The name of the image here is 'Penguins'. Pretty straightforward, right?

- ## Image Shape/Resolution:

We can make use of the shape sub-function to print out the shape of the image. Check out the below image:

```
Import cv2

# Black and White (gray scale)

Img = cv2.imread (&ldquo;Penguins.jpg&rdquo;,0)

Print(img.shape)
```

By shape of the image, we mean the shape of the NumPy array. As you see from executing the code, the matrix consists of 768 rows and 1024 columns.

- ## Displaying the image:

Displaying an image using OpenCV is pretty simple and straightforward. Consider the below image:

```
import cv2

# Black and White (gray scale)

Img = cv2.imread (&ldquo;Penguins.jpg&rdquo;,0)

cv2.imshow(&ldquo;Penguins&rdquo;, img)

cv2.waitKey(0)

# cv2.waitKey(2000)

cv2.destroyAllWindows()
```

As you can see, we first import the image using **imread.** We require a window output to display the images, right?

We use the **imshow** function to display the image by opening a window. There are 2 parameters to the **imshow** function which is the name of the window and the image object to be displayed.

Later, we wait for a user event. **waitKey** makes the window static until the user presses a key. The parameter passed to it is the time in milliseconds.

And lastly, we use **destroyAllWindows** to close the window based on the waitForKey parameter.

- ## Resizing the image:

Similarly, resizing an image is very easy. Here's another code snippet:

```
import cv2

# Black and White (gray scale)

img = cv2.imread (&ldquo;Penguins.jpg&rdquo;,0)

resized_image = cv2.resize(img, (650,500))

cv2.imshow(&ldquo;Penguins&rdquo;, resized_image)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Here, **resize** function is used to resize an image to the desired shape. The parameter here is the shape of the new resized image.

Later, do note that the image object changes from **img** to **resized_image,** because of the image object changes now.

Rest of the code is pretty simple to the previous one, correct?

I am sure you guys are curious to look at the penguins, right? This is the image we were looking to output all this while!

Fig:2.3

There is another way to pass the parameters to the resize function. Check out the following representation:

```
Resized_image = cv2.resize(img, int(img.shape[1]/2), int(img.shape[0]/2)))
```

Here, we get the new image shape to be half of that of the original image.

Next up on this OpenCV Python Tutorial blog, let us look at how we perform face detection using OpenCV.

- # Face Detection Using OpenCV

This seems complex at first but it is very easy. Let me walk you through the entire process and you will feel the same.

- **Step 1:** Considering our prerequisites, we will require an image, to begin with. Later we need to create a cascade classifier which will eventually give us the features of the face.
- **Step 2:** This step involves making use of OpenCV which will read the image and the features file. So at this point, there are NumPy arrays at the primary data points.

All we need to do is to search for the row and column values of the face NumPy ndarray. This is the array with the face rectangle coordinates.

- **Step 3:** This final step involves displaying the image with the rectangular face box.

Check out the following image, here I have summarized the 3 steps in the form of an image for easier readability:

Pretty straightforward, correct?

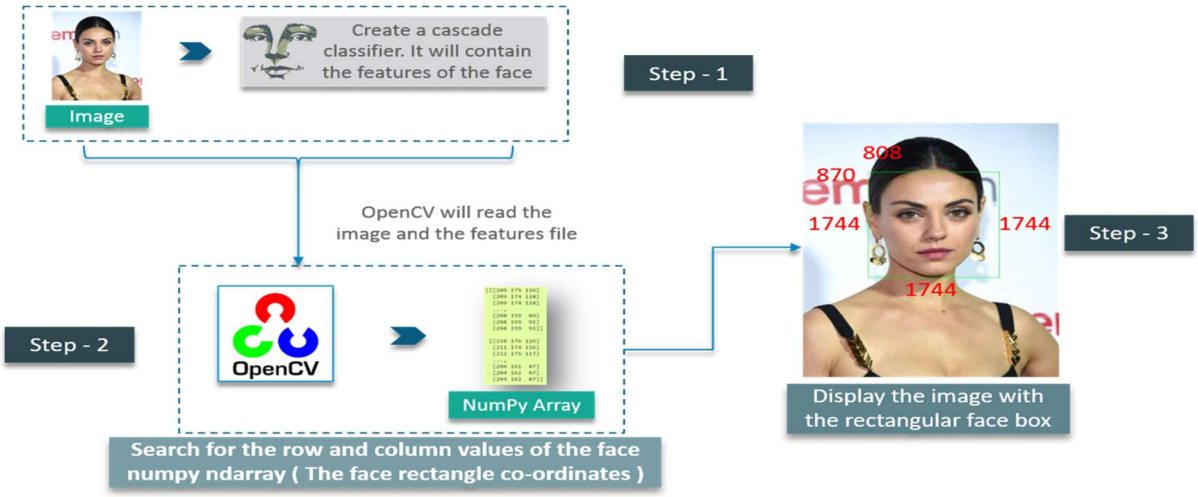


Fig:2.4

First, we create a **CascadeClassifier** object to extract the features of the face as explained earlier. The path to the XML file which contains the face features is the parameter here.

The next step would be to read an image with a face on it and convert it into a black and white image using **COLOR_BGR2GREY**. Followed by this, we search for the coordinates for the image. This is done using **detectMultiScale**.

What coordinates, you ask? It's the coordinates for the face rectangle. The **scaleFactor** is used to decrease the shape value by 5% until the face is found. So, on the whole – Smaller the value, greater is the accuracy.

Finally, the face is printed on the window.

- **Adding the rectangular face box:**

This logic is very simple – As simple as making use of a for loop statement. Check out the following image

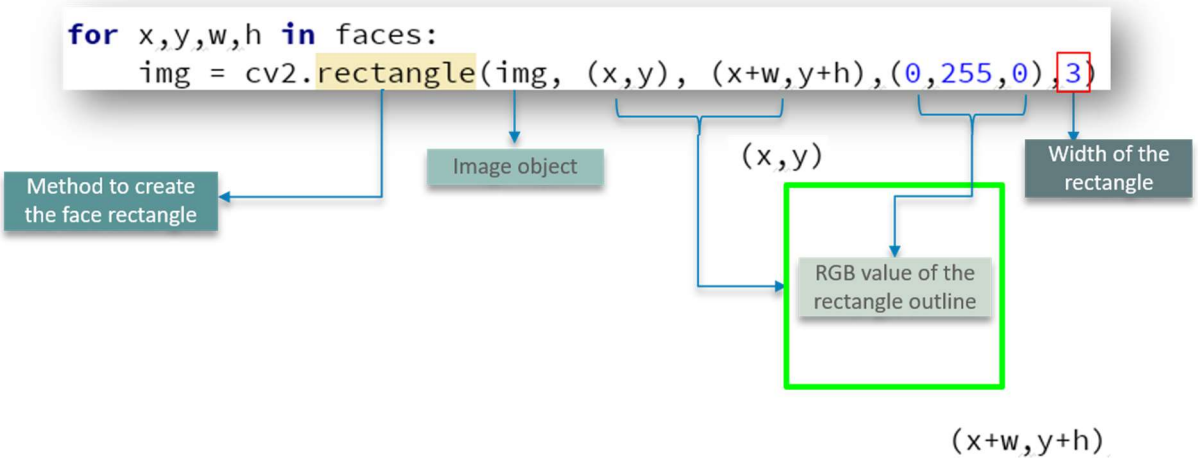


Fig.:2.5

We define the method to create a rectangle using **cv2.rectangle** by passing parameters such as the image object, RGB values of the box outline and the width of the rectangle.Let us check out the entire code for face detection:

```python
import cv2

# Create a CascadeClassifier Object
face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

# Reading the image as it is
img = cv2.imread("photo.jpg")

# Reading the image as gray scale image
gray_img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

# Search the co-ordintes of the image
faces = face_cascade.detectMultiScale(gray_img, scaleFactor = 1.05,
        
             
for x,y,w,h in faces:
    img = cv2.rectangle(img, (x,y), (x+w,y+h),(0,255,0),3)

resized = cv2.resize(img, (int(img.shape[1]/7),int(img.shape[0]/7)))

cv2.imshow("Gray", resized)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Next up on this OpenCV Python Tutorial blog, let us look at how to use OpenCV to capture video with the computer webcam.

• Capturing Video Using OpenCV

Capturing videos using OpenCV is pretty simple as well. the following loop will give you a better idea. Check it out:

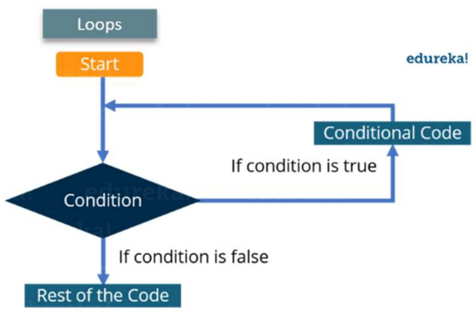


Fig.:2.6

The images are read one-by-one and hence videos are produced due to fast processing of frames which makes the individual images move.

- # Capturing Video:

Check out the following image:

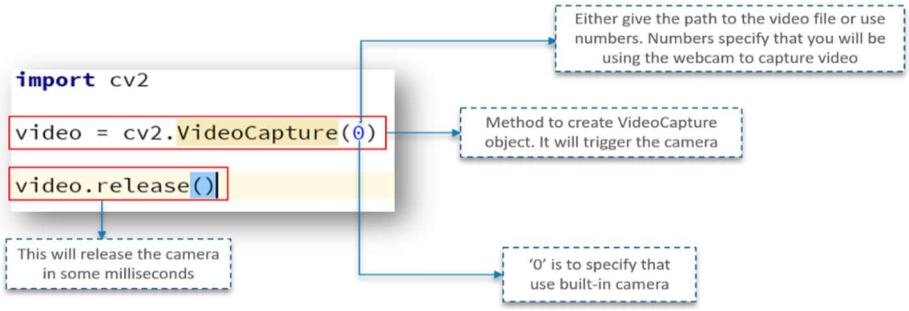


Fig.:2.7

First, we import the OpenCV library as usual. Next, we have a method called **VideoCapture** which is used to create the VideoCapture object. This method is used to trigger the camera on the user's machine. The parameter to this function denotes if the program should make use of the built-in camera or an add-on camera. '0' denotes the built-in camera in this case.

And lastly, the **release** method is used to release the camera in a few milliseconds.When you go ahead and type in and try to execute the above code, you will notice that the camera light switches on for a split second and turns off later. Why does this happen? This happens because there is no time delay to keep the camera functional.

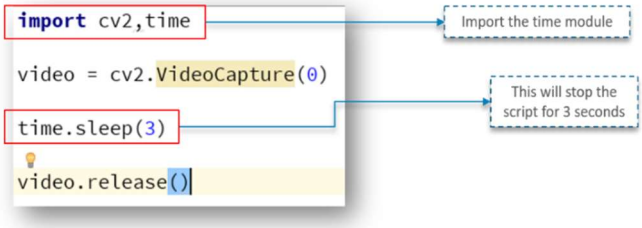


Fig.:2.8

Looking at the above code, we have a new line called **time.sleep(3)** – This makes the script to stop for 3 seconds. Do note that the parameter passed is the time in seconds. So, when the code is executed, the webcam will be turned on for 3 seconds.

- **Adding the window:**

Adding a window to show the video output is pretty simple and can be compared to the same methods used for images. However, there is a slight change. Check out the following code:

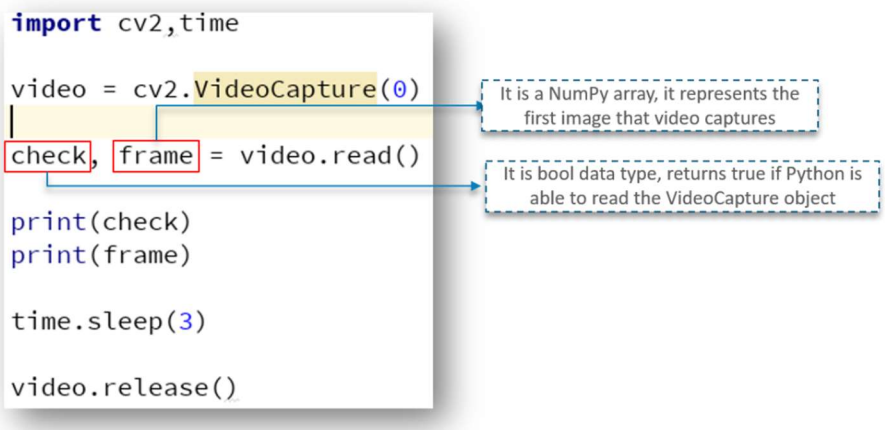


Fig.:2.9

Here, we have defined a NumPy array which we use to represent the first image that the video captures – This is stored in the **frame** array.

We also have **check** – This is a boolean datatype which returns **True** if Python is able to access and read the **VideoCapture** object.

Check out the output below:

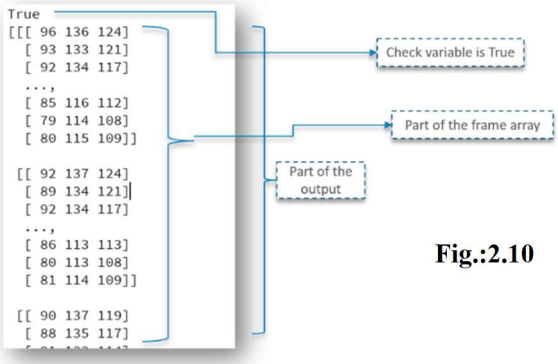


Fig.:2.10

As you can check out, we got the output as **True** and the part of the frame array is printed.

But we need to read the first frame/image of the video to begin, correct?

To do exactly that, we need to first create a frame object which will read the images of the **VideoCapture** object.

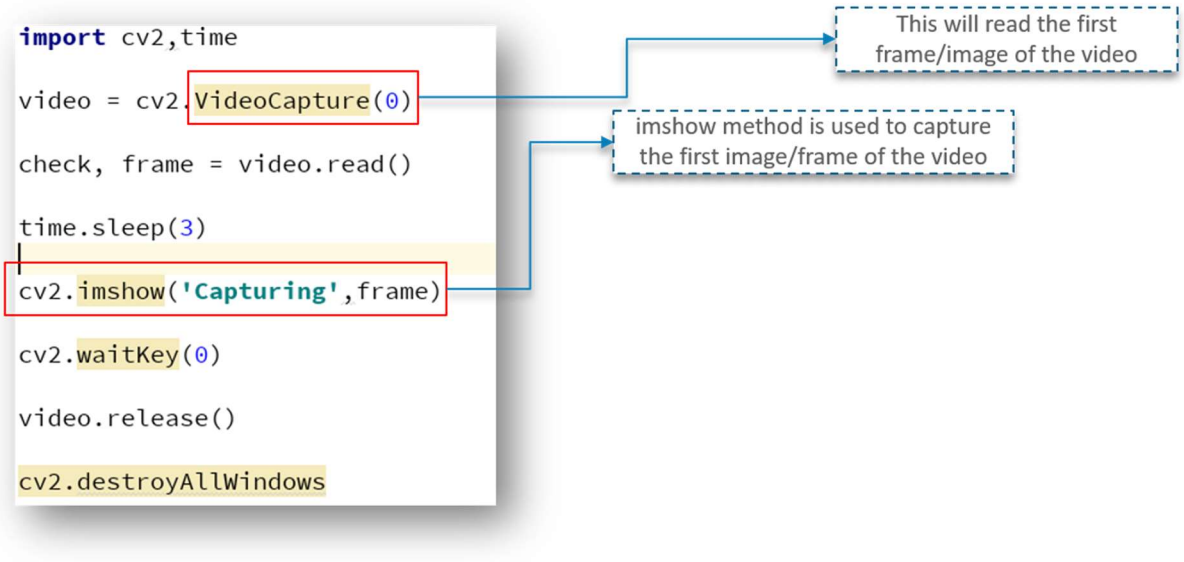


Fig.:2.11

As seen above, the **imshow** method is used to capture the first frame of the video.

All this while, we have tried to capture the first image/frame of the video but directly capturing the video.

- **Capturing Video Directly:**

In order to capture the video, we will be using the **while** loop. While condition will be such that, until unless **'check'** is **True.** If it is, then Python will display the frames.

Here's the code snippet image:

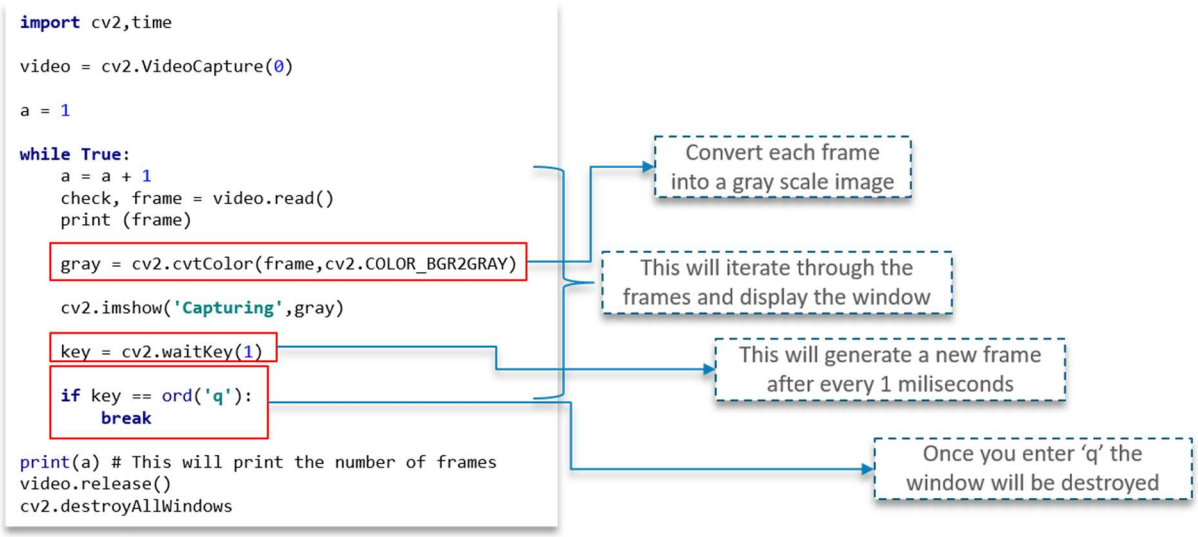


Fig.:2.12

We make use of the **cvtColor** function to convert each frame into a grey-scale image as explained earlier.

waitKey(1) will make sure to generate a new frame after every millisecond of a gap.

It is important here that you note that the **while** loop is completely in play to help iterate through the frames and eventually display the video.

There is a user event trigger here as well. Once the 'q' key is pressed by the user, the program window closes.

OpenCV is pretty easy to grasp, right? I personally love how good the readability is and how quickly a beginner can get started working with OpenCV.

Next up on this OpenCV Python Tutorial blog, let us look at how to use a very interesting motion detector use case using OpenCV.

- # Use Case: Motion Detector Using OpenCV
- # Problem Statement:

You have been approached by a company that is studying human behavior. Your task is to give them a webcam, that can detect the motion or any movement in front of it. This should

return a graph, this graph should contain how long the human/object was in front of the camera.

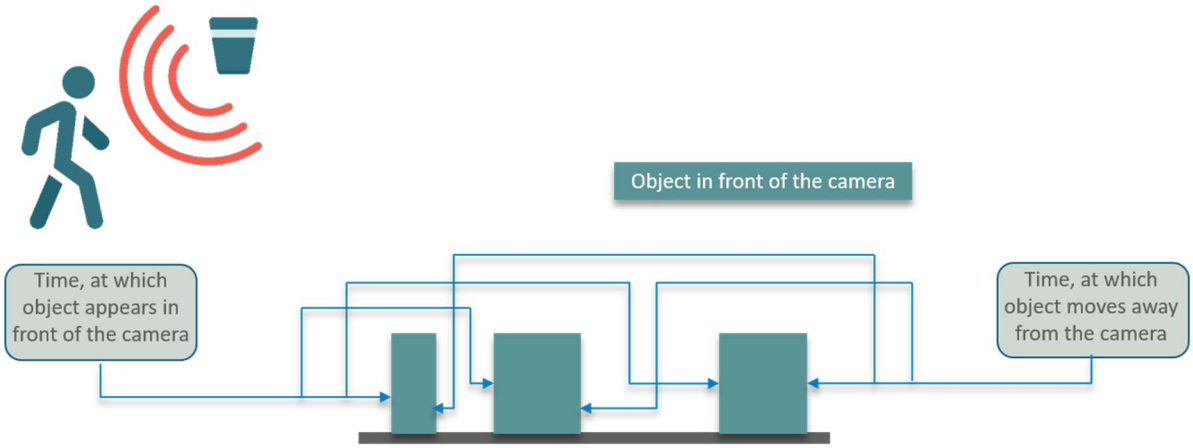


Fig.:2.13

So, now that we have defined our problem statement, we need to build a solution logic to approach the problem in a structured way.

Consider the below diagram:

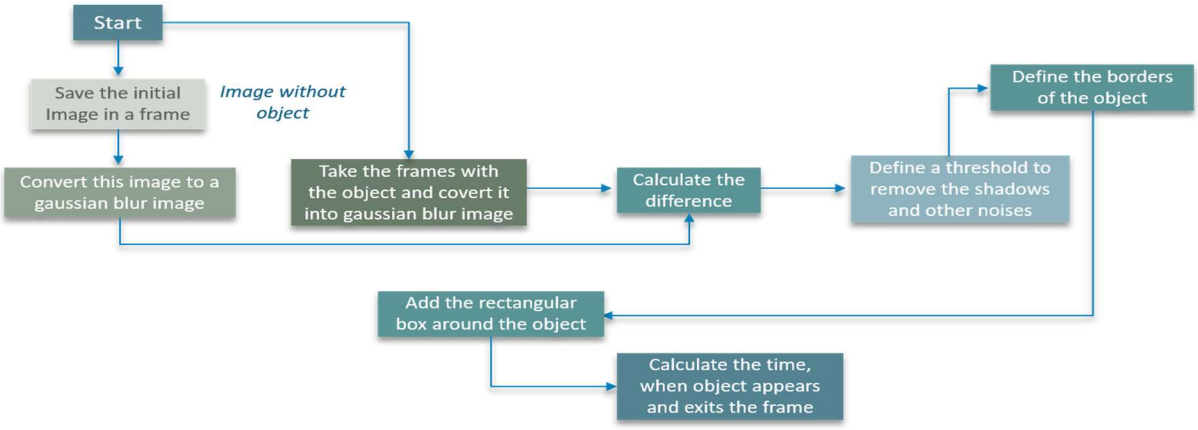


Fig.:2.14

Initially, we save the image in a particular frame.

The next step involves converting the image to a Gaussian blur image. This is done so as to ensure we calculate a palpable difference between the blurred image and the actual image.

At this point, the image is still not an object. We define a threshold to remove blemishes such as shadows and other noises in the image.

Borders for the object are defined later and we add a rectangular box around the object as we discussed earlier on the blog.

Lastly, we calculate the time at which the object appears and exits the frame.

Pretty easy, right?

Here's the code snippet:

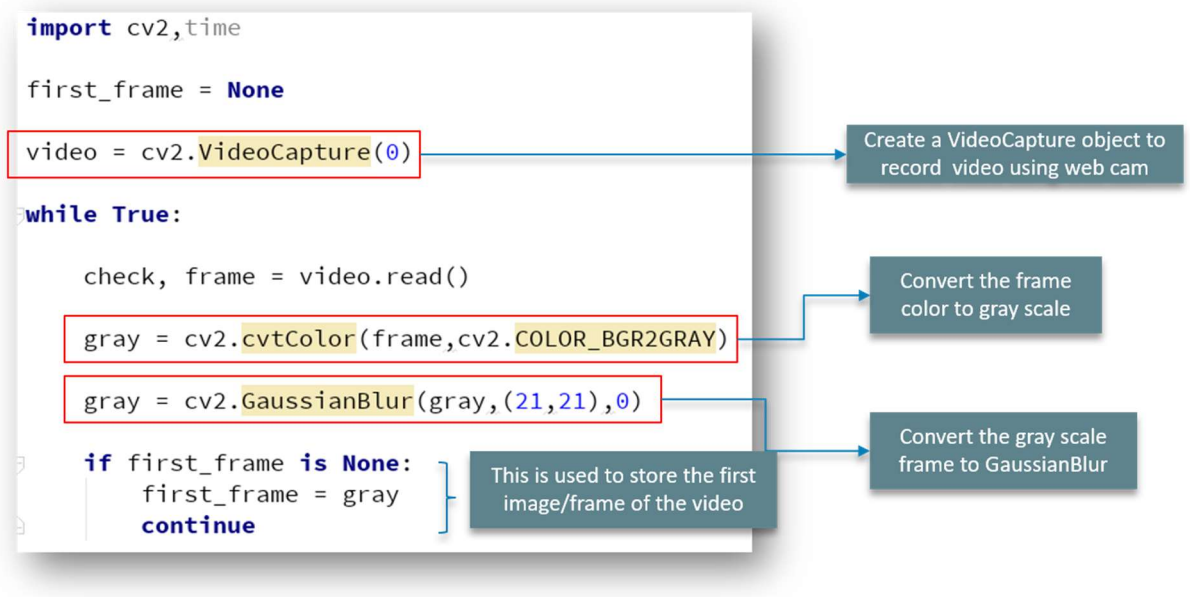


Fig.:2.15

The same principle follows through here as well. We first import the package and create the **VideoCapture** object to ensure we capture video using the webcam.

The while loop iterates through the individual frames of the video. We convert the color frame to a grey-scale image and later we convert this grey-scale image to Gaussian blur.

We need to store the first image/frame of the video, correct? We make use of the **if** statement for this purpose alone.

Now, let us dive into a little more code:

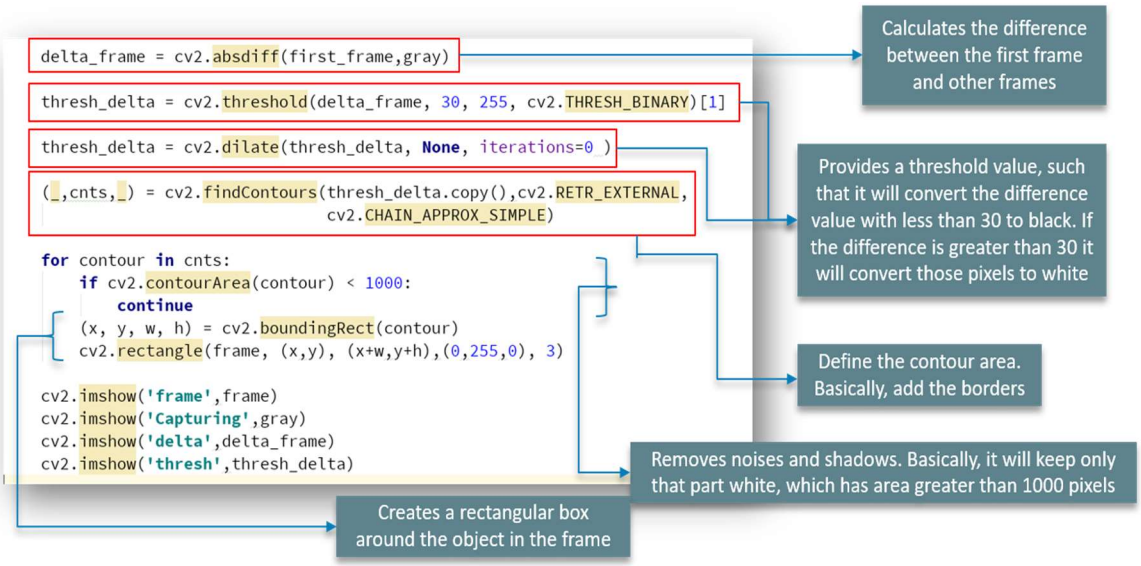


Fig.:2.16

We make use of the **absdiff** function to calculate the difference between the first occurring frame and all the other frames.

The **threshold** function provides a threshold value, such that it will convert the difference value with less than 30 to black. If the difference is greater than 30 it will convert those pixels to white color. **THRESH_BINARY** is used for this purpose.

Later, we make use of the **findContours** function to define the contour area for our image. And we add in the borders at this stage as well.

The **contourArea** function, as previously explained, removes the noises and the shadows. To make it simple, it will keep only that part white, which has an area greater than 1000 pixels as we've defined for that.

Later, we create a rectangular box around our object in the working frame.

And followed by this is this simple code:

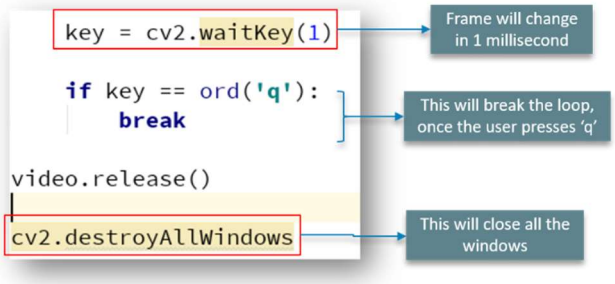


Fig.2.17

As discussed earlier, the frame changes every 1 millisecond and when the user enters **'q',** the loop breaks and the window closes.

We've covered all of the major details on this OpenCV Python Tutorial blog. One thing that remains with our use-case is that we need to calculate the time for which the object was in front of the camera.

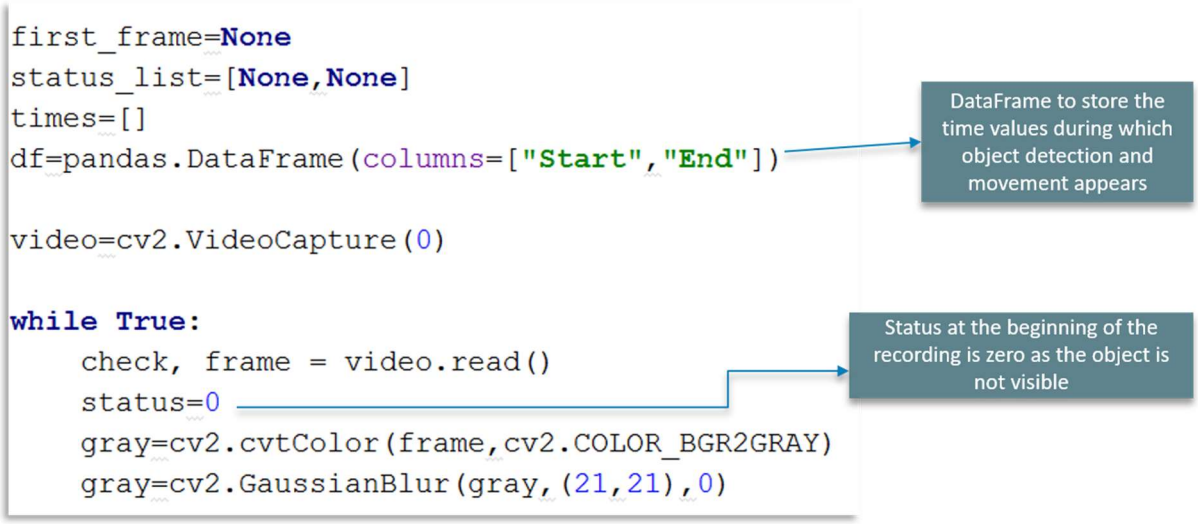


Fig.2.18

We make use of **DataFrame** to store the time values during which object detection and movement appear in the frame.

Followed by that is **VideoCapture** function as explained earlier. But here, we have a flag bit we call **status.** We use this status at the beginning of the recording to be **zero** as the object is not **visible** initially.

```
(_,cnts,_)=cv2.findContours(thresh_frame.copy(),cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

for contour in cnts:
    if cv2.contourArea(contour) < 10000:
        continue
    status=1
```

Change in status when the object is being detected

Fig.2.19

We will change the status flag to 1 when the object is being detected as shown in the above figure. Pretty simple, right?

```
    (x, y, w, h)=cv2.boundingRect(contour)
    cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,0), 3)
status_list.append(status)

status_list=status_list[-2:]


if status_list[-1]==1 and status_list[-2]==0:
    times.append(datetime.now())
if status_list[-1]==0 and status_list[-2]==1:
    times.append(datetime.now())
```

List of status for every frame

Record datetime in a list when change occurs

Fig.2.20

We are going to make a list of the status for every scanned frame and later record the date and time using **datetime** in a list if and where a change occurs.

```
print(status_list)
print(times)

for i in range(0,len(times),2):
    df=df.append({"Start":times[i],"End":times[i+1]},ignore_index=True)

df.to_csv("Times.csv")

video.release()
cv2.destroyAllWindows
```

Store time values in a DataFrame

Write the DataFrame to a CSV file

Fig.2.21

And we store the time values in a **DataFrame** as shown in the above explanatory diagram. We'll conclude by writing the **DataFrame** to a CSV file as shown.

Plotting the Motion Detection Graph:

The final step in our use-case to display the results. We are displaying the graph which denotes the motion on 2-axes. Consider the below code:

```
from motion_detector import df
from bokeh.plotting import figure, show, output_file
from bokeh.models import HoverTool, ColumnDataSource

df["Start_string"]=df["Start"].dt.strftime("%Y-%m-%d %H:%M:%S")
df["End_string"]=df["End"].dt.strftime("%Y-%m-%d %H:%M:%S")


cds=ColumnDataSource(df)

p=figure(x_axis_type='datetime',height=100, width=500, responsive=True,title="Motion Graph")
p.yaxis.minor_tick_line_color=None
p.ygrid[0].ticker.desired_num_ticks=1

hover=HoverTool(tooltips=[("Start","@Start_string"),("End","@End_string")])
p.add_tools(hover)

q=p.quad(left="Start",right="End",bottom=0,top=1,color="red",source=cds)

output_file("Graph1.html")
show(p)
```

Import the DataFrame from the motion_detector.py

Convert time to a string format

The DataFrame of time values is plotted on the browser using Bokeh plots

Fig.2.22

To begin with, we import the **DataFrame** from the **motion_detector.py** file.

The next step involves converting time to a readable string format which can be parsed.

Lastly, the **DataFrame** of time values is plotted on the browser using **Bokeh plots.**

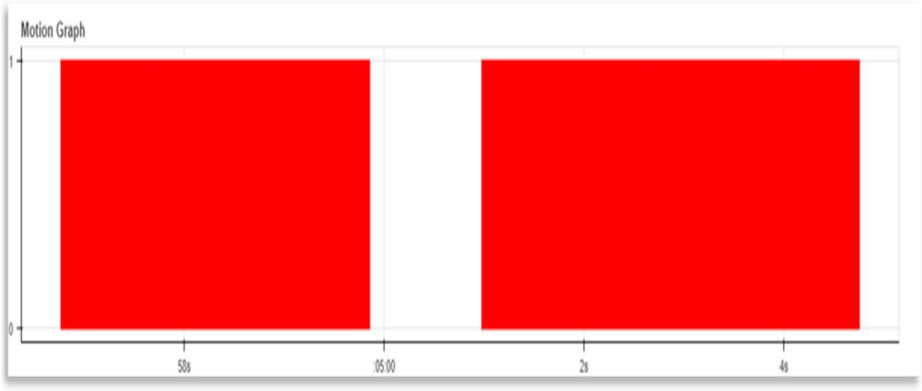
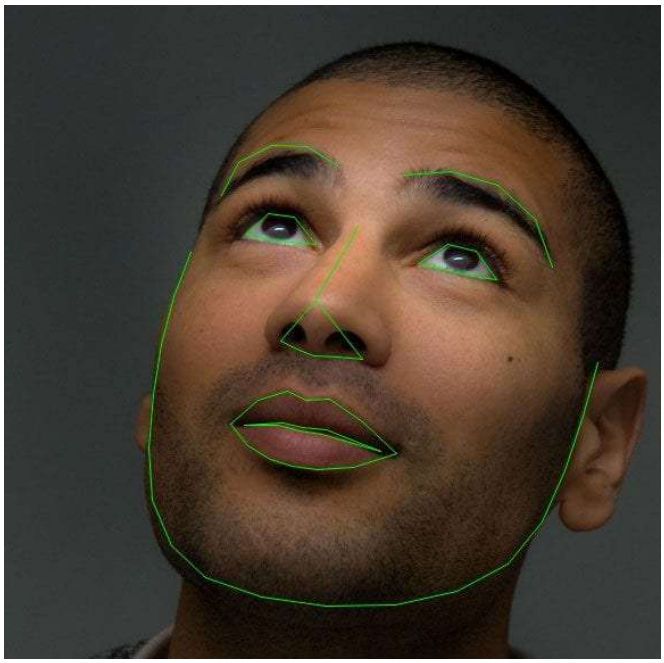


Fig.2.23

This will be very handy when you are trying to develop applications that require image recognition and similar principles. Now, you should also be able to use these concepts to develop applications easily with the help of OpenCV in Python.

2.1.2 Dlib 68 point & Facial recognition:

- **what are facial landmarks?**



Detecting facial landmarks is a subset of the shape prediction problem. Given an input image (and normally an ROI that specifies the object of interest), a shape predictor attempts to localize key points of interest along the shape.

In the context of facial landmarks, our goal is detect important facial structures on the face using shape prediction methods.

Detecting facial landmarks is therefore a two step process:

Step #1: Localize the face in the image.
Step #2: Detect the key facial structures on the face ROI.
Face detection (Step #1) can be achieved in a number of ways.

We could use OpenCV's built-in Haar cascades.

We might apply a pre-trained HOG + Linear SVM object detector specifically for the task of face detection.

Or we might even use deep learning-based algorithms for face localization.

In either case, the actual algorithm used to detect the face in the image doesn't matter. Instead, what's important is that through some method we obtain the face bounding box (i.e., the (x, y)-coordinates of the face in the image).

Given the face region we can then apply Step #2: detecting key facial structures in the face region.

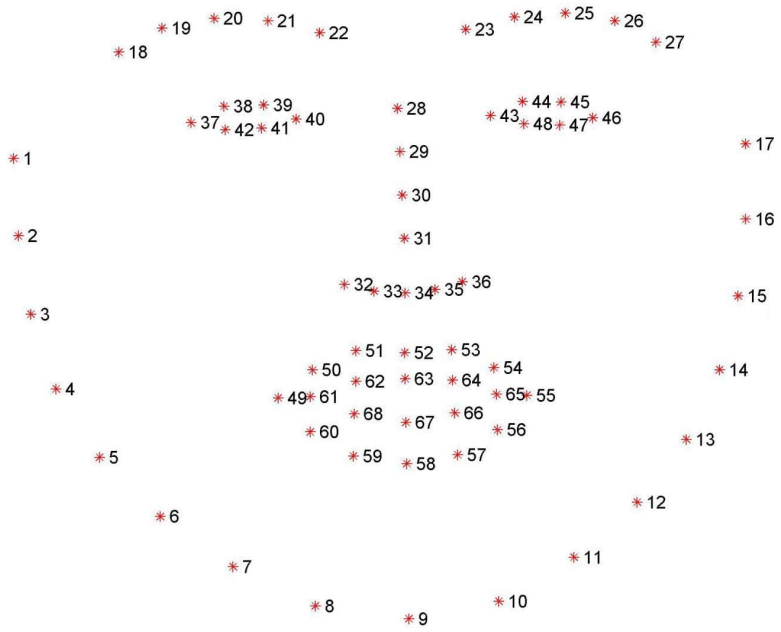
There are a variety of facial landmark detectors, but all methods essentially try to localize and label the following facial regions:

Mouth
Right eyebrow
Left eyebrow
Right eye
Left eye
Nose
Jaw
The facial landmark detector included in the dlib library is an implementation of the One Millisecond Face Alignment with an Ensemble of Regression Trees paper by Kazemi and Sullivan (2014).

This method starts by using:

- A training set of labeled facial landmarks on an image. These images are manually labeled, specifying specific (x, y)-coordinates of regions surrounding each facial structure.
- Priors, of more specifically, the probability on distance between pairs of input pixels. Given this training data, an ensemble of regression trees are trained to estimate the facial landmark positions directly from the pixel intensities themselves (i.e., no "feature extraction" is taking place).

The end result is a facial landmark detector that can be used to detect facial landmarks in real-time with high quality predictions.



These annotations are part of the 68 point iBUG 300-W dataset which the dlib facial landmark predictor was trained on.

It's important to note that other flavors of facial landmark detectors exist, including the 194 point model that can be trained on the HELEN dataset.

Regardless of which dataset is used, the same dlib framework can be leveraged to train a shape predictor on the input training data — this is useful if you would like to train facial landmark detectors or custom shape predictors of your own.

In the remaining of this blog post I'll demonstrate how to detect these facial landmarks in images.

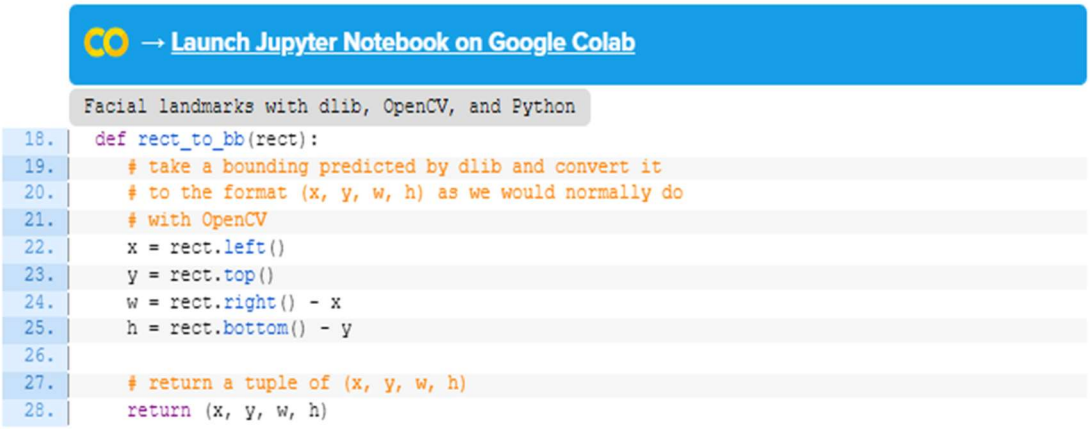
Future blog posts in this series will use these facial landmarks to extract specific regions of the face, apply face alignment, and even build a blink detection system.

- **Detecting facial landmarks with dlib, OpenCV, and Python**

In order to prepare for this series of blog posts on facial landmarks, I've added a few convenience functions to my imutils library, specifically inside **face_utils.py.**

We'll be reviewing two of these functions inside face_utils.py now and the remaining ones next week.

The first utility function is rect_to_bb , short for "rectangle to bounding box":



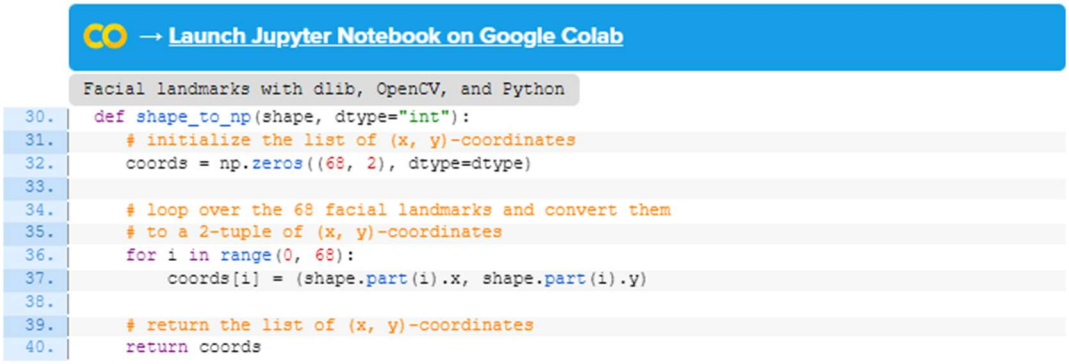
This function accepts a single argument, rect, which is assumed to be a bounding box rectangle produced by a dlib detector (i.e., the face detector).

The rect object includes the (x, y)-coordinates of the detection.

However, in OpenCV, we normally think of a bounding box in terms of "(x, y, width, height)" so as a matter of convenience, the rect_to_bb function takes this rect object and transforms it into a 4-tuple of coordinates.

Again, this is simply a matter of conveinence and taste.

Secondly, we have the shape_to_np function:



The dlib face landmark detector will return a shape object containing the 68 (x, y)-coordinates of the facial landmark regions.

Using the shape_to_np function, we cam convert this object to a NumPy array, allowing it to "play nicer" with our Python code.

Given these two helper functions, we are now ready to detect facial landmarks in images.

Open up a new file, name it facial_landmarks.py , and insert the following code:

→ Launch Jupyter Notebook on Google Colab

Facial landmarks with dlib, OpenCV, and Python

```
1. # import the necessary packages
2. from imutils import face_utils
3. import numpy as np
4. import argparse
5. import imutils
6. import dlib
7. import cv2
8.
9. # construct the argument parser and parse the arguments
10. ap = argparse.ArgumentParser()
11. ap.add_argument("-p", "--shape-predictor", required=True,
12.    help="path to facial landmark predictor")
13. ap.add_argument("-i", "--image", required=True,
14.    help="path to input image")
15. args = vars(ap.parse_args())
```

Lines 2-7 import our required Python packages.

We'll be using the face_utils submodule of imutils to access our helper functions detailed above.

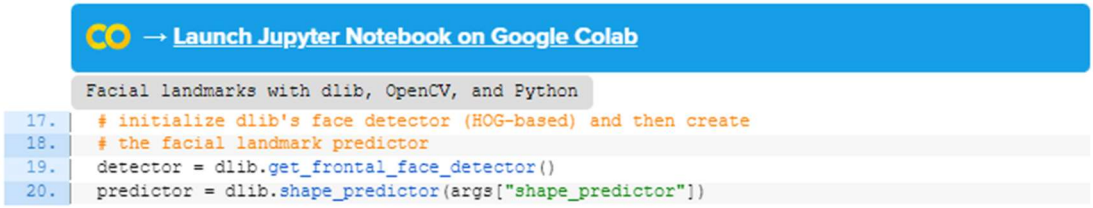
We'll then import dlib . If you don't already have dlib installed on your system, please follow the instructions in my previous blog post to get your system properly configured.

Lines 10-15 parse our command line arguments:

--shape-predictor : This is the path to dlib's pre-trained facial landmark detector. You can download the detector model here or you can use the "Downloads" section of this post to grab the code + example images + pre-trained detector as well.

--image : The path to the input image that we want to detect facial landmarks on.

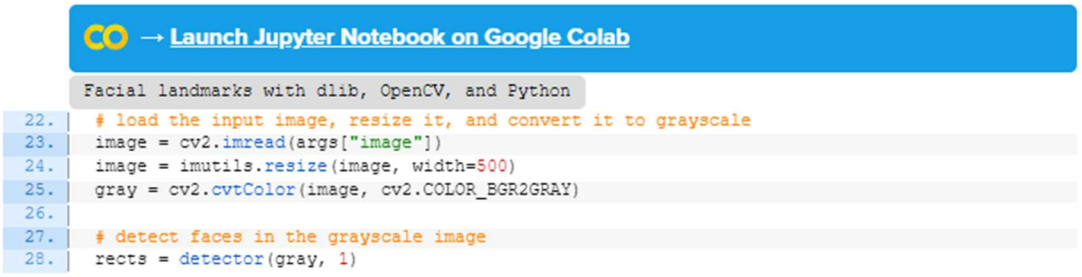
Now that our imports and command line arguments are taken care of, let's initialize dlib's face detector and facial landmark predictor:



Line 19 initializes dlib's pre-trained face detector based on a modification to the standard Histogram of Oriented Gradients + Linear SVM method for object detection.

Line 20 then loads the facial landmark predictor using the path to the supplied --shape-predictor.

But before we can actually detect facial landmarks, we first need to detect the face in our input image:



Line 23 loads our input image from disk via OpenCV, then pre-processes the image by resizing to have a width of 500 pixels and converting it to grayscale (Lines 24 and 25).

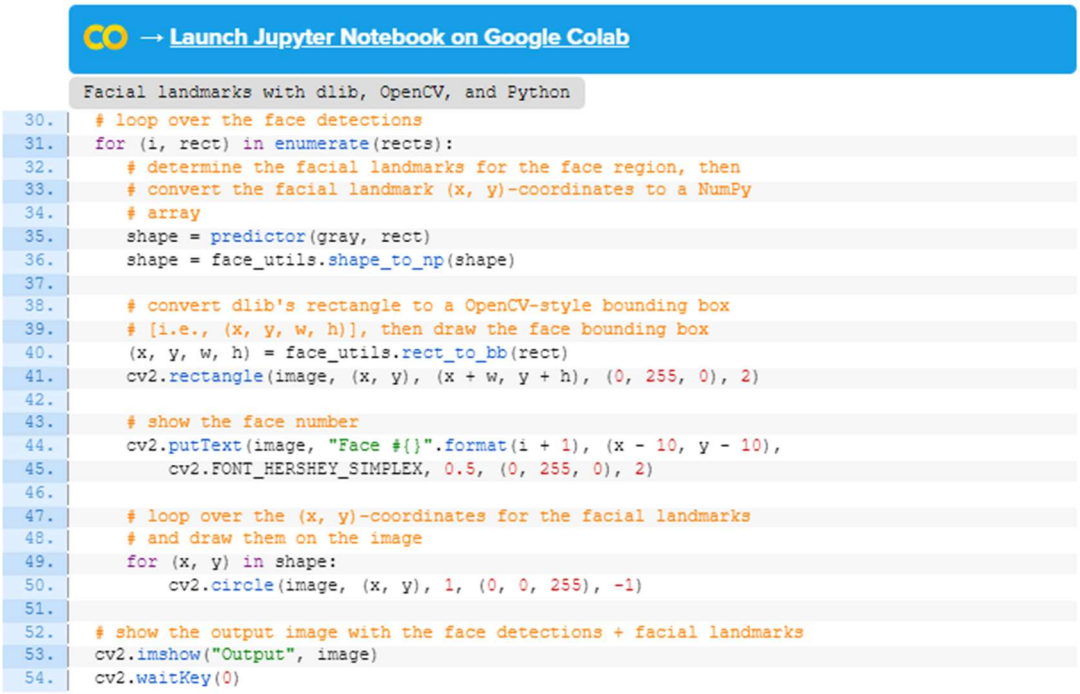
Line 28 handles detecting the bounding box of faces in our image.

The first parameter to the detector is our grayscale image (although this method can work with color images as well).

The second parameter is the number of image pyramid layers to apply when upscaling the image prior to applying the detector (this it the equivalent of computing cv2.pyrUp N number of times on the image).

The benefit of increasing the resolution of the input image prior to face detection is that it may allow us to detect more faces in the image — the downside is that the larger the input image, the more computaitonally expensive the detection process is.

Given the (x, y)-coordinates of the faces in the image, we can now apply facial landmark detection to each of the face regions:



We start looping over each of the face detections on Line 31.

For each of the face detections, we apply facial landmark detection on Line 35, giving us the 68 (x, y)-coordinates that map to the specific facial features in the image.

Line 36 then converts the dlib shape object to a NumPy array with shape (68, 2).

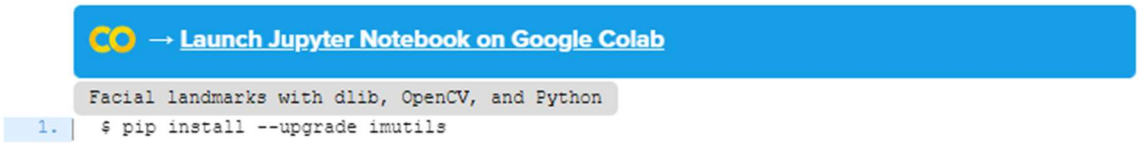
Lines 40 and 41 draw the bounding box surrounding the detected face on the image while Lines 44 and 45 draw the index of the face.

Finally, Lines 49 and 50 loop over the detected facial landmarks and draw each of them individually.

Lines 53 and 54 simply display the output image to our screen.

Facial landmark visualizations

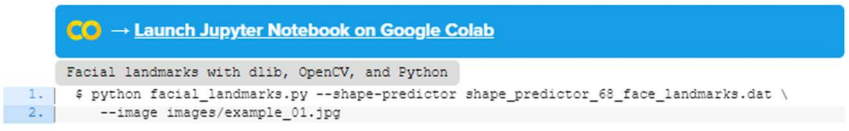
Before we test our facial landmark detector, make sure you have upgraded to the latest version of imutils which includes the face_utils.py file:

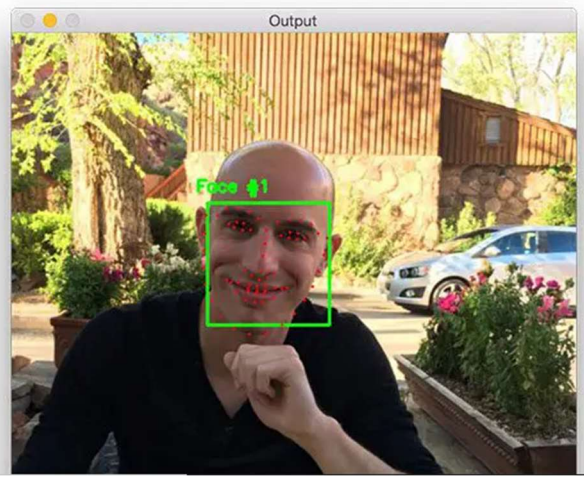


Note: If you are using Python virtual environments, make sure you upgrade the imutils inside the virtual environment.

From there, use the “Downloads” section of this guide to download the source code, example images, and pre-trained dlib facial landmark detector.

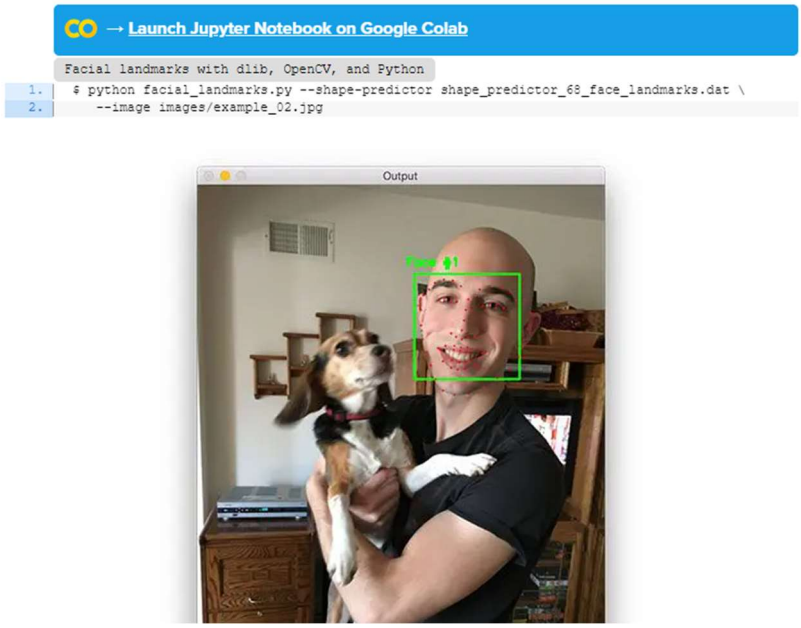
Once you’ve downloaded the .zip archive, unzip it, change directory to facial-landmarks , and execute the following command:





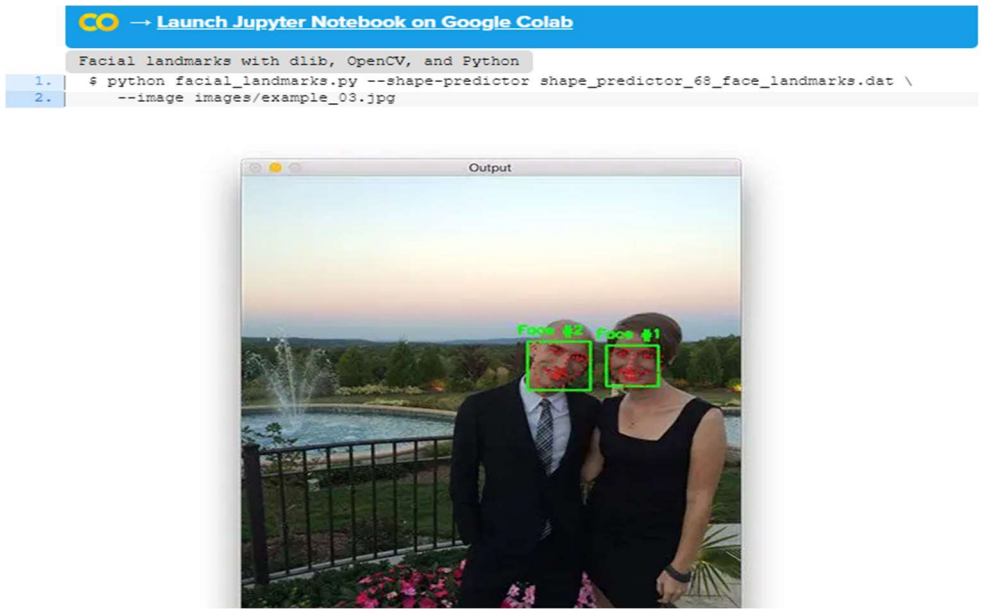
Notice how the bounding box of my face is drawn in green while each of the individual facial landmarks are drawn in red.

The same is true for this second example image:



Here we can clearly see that the red circles map to specific facial features, including my jawline, mouth, nose, eyes, and eyebrows.

Let's take a look at one final example, this time with multiple people in the image:



For both people in the image (myself and Trisha, my fiancée), our faces are not only detected but also annotated via facial landmarks as well.

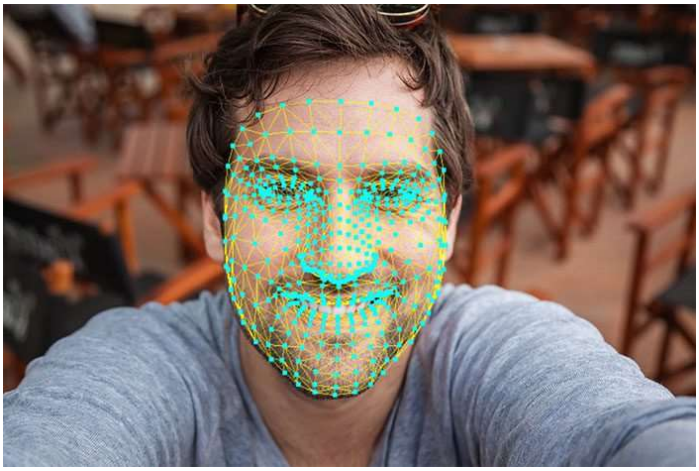
Alternative facial landmark detectors

Dlib's 68-point facial landmark detector tends to be the most popular facial landmark detector in the computer vision field due to the speed and reliability of the dlib library.

However, other facial landmark detection models exist.

To start, dlib provides an alternative 5-point facial landmark detector that is faster than the 68-point variant. This model works great if all you need are the locations of the eyes and the nose.

One of the most popular new facial landmark detectors comes from the MediaPipe library which is capable of computing a 3D face mesh



2.1.3 Extract and warp triangles

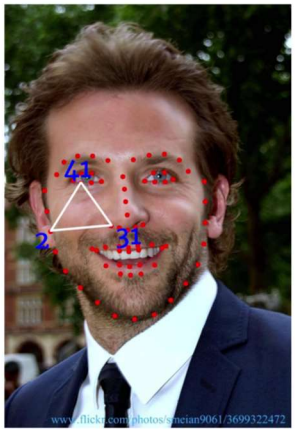
Extract and warp triangles

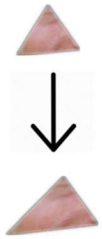
Once we have the triangulation of both faces we take the triangles of the source face and we extract them.

We also need to take the coordinates of the triangles of the destination face, so that we can warp the triangles of the source face to have same size and perspective of the matchin triangle on the destination face.

The code below shows how to warp the triangles of the source image.

```
# Warp triangles
points = np.float32(points)
points2 = np.float32(points2)
M = cv2.getAffineTransform(points, points2)
warped_triangle = cv2.warpAffine(cropped_triangle, M, (w, h))
warped_triangle = cv2.bitwise_and(warped_triangle, warped_triangle, mask=cropped_tr2_mask)
```



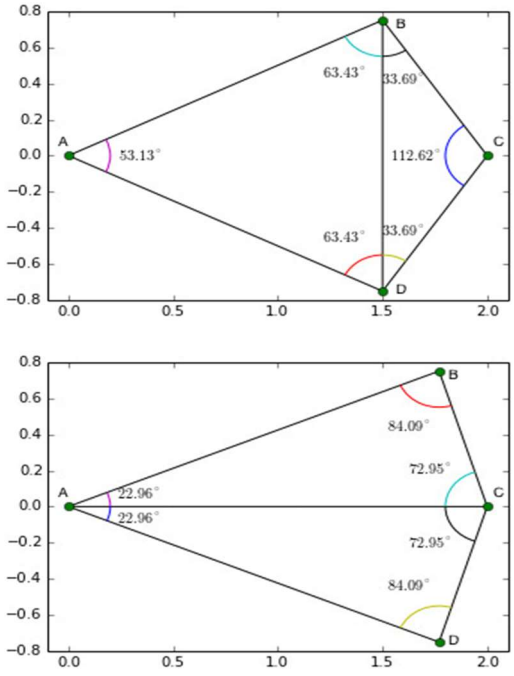


- **What is Delaunay Triangulation ?**

Given a set of points in a plane, a triangulation refers to the subdivision of the plane into triangles, with the points as vertices. In Figure 1, we see a set of landmarks on the left image, and the triangulation in the middle image. A set of points can have many possible triangulations, but Delaunay triangulation stands out because it has some nice properties. In a Delaunay triangulation, triangles are chosen such that no point is inside the circumcircle of any triangle. Figure 2. shows Delaunay triangulation of 4 points A, B, C and D. In the top image, for the triangulation to be a valid Delaunay triangulation, point C should be outside the circumcircle of triangle ABD, and point A should be outside the circumcircle of triangle BCD.

An interesting property of Delaunay triangulation is that it does not favor “skinny” triangles (i.e. triangles with one large angle).

Figure 2 shows how the triangulation changes to pick “fat” triangles when the points are moved. In the top image, the points B and D have their x-coordinates at x = 1.5, and in the bottom image they are moved to the right to x = 1.75. In the top image angles ABC and ABD are large, and Delaunay triangulation creates an edge between B and D splitting the two large angles into smaller angles ABD, ADB, CDB, and CBD. On the other hand in the bottom image, the angle BCD is too large, and Delaunay triangulation creates an edge AC to divide the large angle.



There are many algorithms to find the Delaunay triangulation of a set of points. The most obvious (but not the most efficient) one is to start with any triangulation, and check if the circumcircle of any triangle contains another point. If it does, flip the edges (as show in Figure 2.) and continue until there are no triangles whose circumcircle contains a point.

Any discussion on Delaunay triangulation has to include Voronoi diagrams because the Voronoi diagram of a set of points is mathematical dual to its Delaunay triangulation.

- **What is a Voronoi Diagram ?**

Given a set of points in a plane, a Voronoi diagram partitions the space such that the boundary lines are equidistant from neighboring points. Figure 3. shows an example of a Voronoi diagram calculated from the points shown as black dots. You will notice that every boundary line passes through the center of two points. If you connect the points in neighboring Voronoi regions, you get a Delaunay triangulation!

Delaunay triangulation and Voronoi diagram are related in more ways than one. Georgy Voronoy, the mathematician after which Voronoi diagram is named, was Boris Delaunay's Ph.D. advisor

- **Delaunay Triangulation and Voronoi Diagram using OpenCV**

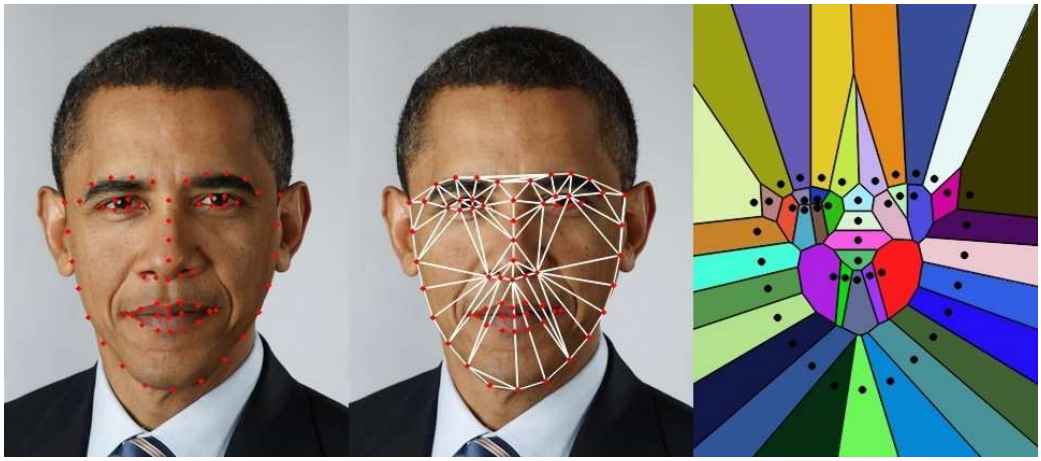


Figure 1. Left : Image of President Obama with landmarks detected using dlib.

Center : Delaunay triangulation of the landmarks. Right : Corresponding Voronoi Diagram.

In a previous post I had discussed two libraries for facial landmark detection, and had pointed to several interesting applications like Face Morphing, Face Replacement etc. that use facial landmarks. In many such applications a triangulation of facial landmarks is first found (See Figure 1), and these triangles are warped to do something interesting. This post will help us understand Delaunay triangulation and Voronoi diagrams (a.k.a Voronoi tesselation, Voronoi decomposition, Voronoi partition, and Dirichlet tessellation), and help us uncover a barely documented function in OpenCV.

- **Delaunay Triangulation & Voronoi Diagram in OpenCV**

- 1. Collect all the points in a vector.

C++

```
1    vector<Point2f>
```

```
points;
//
This
is how
you
can
add
one
point.
points.push_back(Point2f(x,y));
```

- Python

```
points = []
# This
is how
you
can
add
one
point.
points.append((x,
y))
```

- 2. Define the space you want to partition using a rectangle (**rect**). If the points you have defined in the previous step are defined on an image, this rectangle can be (0, 0, width, height). Otherwise, you can choose a rectangle that bounds all the points.

- C++

```
Mat img =
imread("image.jpg");
Size size =
img.size();
Rect rect(0,
0,
size.width,
size.height);
```

- Python

```
img = cv2.imread("image.jpg");
size = img.shape
rect = (0, 0,
size[1],
size[0])
```

- Create an instance of Subdiv2D with the rectangle obtained in the previous step.

  C++

```
Subdiv2D
subdiv(rect);
```

- Python

```
subdiv  =
cv2.Subdiv2D(rect);
```

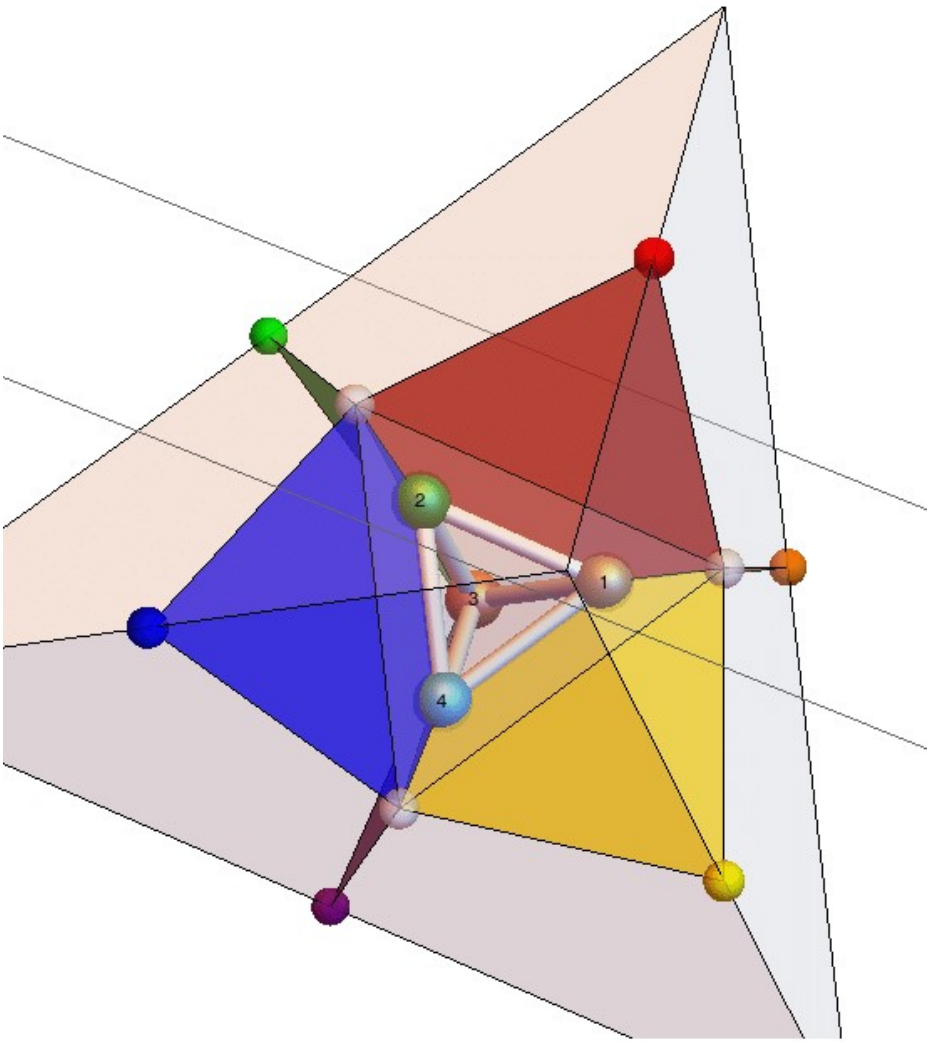
Insert the points into **subdiv** using **subdiv.insert(point)**. The video above shows an animation of triangulation as we add points to subdiv.Use **subdiv.getTriangleList** to get a list of Delaunay triangles.Use **subdiv.getVoronoiFacetList** to get the list of Voronoi facets.

- # OpenCV Example for Delaunay Triangulation & Voronoi Diagram

Here is a complete working example. I have copied some of this code from examples that come with OpenCV and simplified and modified it to suit our purpose. The python example that comes with OpenCV uses the old (and ugly) interface, and so I wrote it from scratch. This code assumes an image is stored in **image.jpg** and the points are stored in **points.txt**. Each row of **points.txt** contains the x and y coordinates of a point separated by a space. E.g.

207 242
210 269
214 297
220 322
229 349

- **Triangulation of images for Face Mesh Application**



first, we need to pass the image to the detector then that object will be used to extract landmarks using predictor. Afterward, storing extracted landmarks (x and y) into the landmarks (list). We're going to mesh up the face into triangles. This step is the core of our face-swapping. Here we will interchange each triangle with the correspondent triangle of the destination image. The triangulation of the destination image needs to have the same pattern just like that of the triangulation of the source image. This means that the connection of the connecting symbols has to be the same. So after we do the triangulation of the source image, from that triangulation we take the indexes of the (x and y) so that we can replicate the same triangulation on the destination image. Once we have the triangles indexes we loop through them and we triangulate the destination face.

2.1.4 convex_hull method:

- **What is a Convex Hull?**

  Let us break the term down into its two parts — Convex and Hull.

  A Convex object is one with no interior angles greater than 180 degrees. A shape that is not convex is called Non-Convex or Concave.

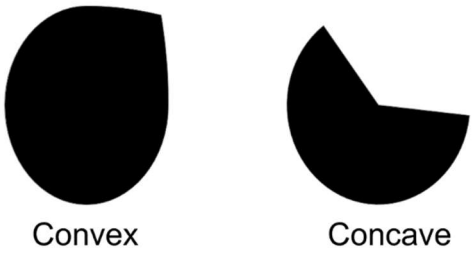


Figure 1: Example of a Convex Object and a Concave Object

Hull means the exterior or the shape of the object.

Therefore, the Convex Hull of a shape or a group of points is a tight fitting convex boundary around the points or the shape.

The Convex Hull of the two shapes in Figure 1 is shown in Figure 2. The Convex Hull of a convex object is simply its boundary. The Convex Hull of a concave shape is a convex boundary that most tightly encloses it

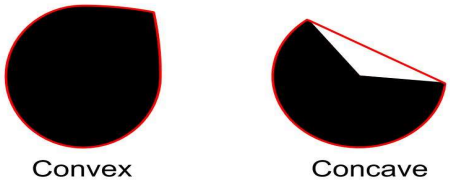


Figure 2: The Convex hull of the two black shapes is shown in red.

- **Gift Wrapping Algorithms**

Given a set of points that define a shape, how do we find its convex hull? The algorithms for finding the Convext Hull are often called **Gift Wrapping** algorithms. The video below explains a few algorithms with excellent animations:

Easy, huh? A lot of things look easy on the surface, but as soon as we impose certain constraints on them, things become pretty hard.

For example, the Jarvis March algorithm described in the video has complexity O(nh) where n is the number of input points and h is the number of points in the convex hull. Chan's algorithm has complexity O(n log h).

Is an O(n) algorithm possible? The answer is YES, but boy the history of finding a linear algorithm for convex hull is a tad embrassing.

The first O(n) algorithm was published by Sklansky in 1972. It was later proven to be incorrect. Between 1972 and 1989, 16 different linear algorithms were published and 7 of them were found to be incorrect later on!

This reminds me of a joke I heard in college. Every difficult problem in mathematics has a simple, easy to understand wrong solution!

Now, here is an embarrassing icing on the embarrassing cake. The algorithm implemented in OpenCV is one by Sklansky (1982). It happens to be INCORRECT!

It is still a popular algorithm and in a vast majority of cases, it produces the right result. This algorithm is implemented in the convexHull class in OpenCV. Let's now see how to use it.

- **convexHull in OpenCV**

  By now we know the Gift Wrapping algorithms for finding the Convex Hull work on a collection of points.

  How do we use it on images?

  We first need to binarize the image we are working with, find contours and finally find the convex hull. Let's go step by step.

  Step 1: Read the Input Image

Python

```
src = cv2.imread("sample.jpg", 1) # read input image
```

C++

```
Mat src;
src = imread("sample.jpg", 1); // read input image
```

Step 2: Binarize the input image

We perform binarization in three steps —

1. Convert to grayscale
2. Blur to remove noise
3. Threshold to binarize image

The results of these steps are shown in Figure 3. And here is the code.

Python

```
gray = cv2.cvtColor(src,
cv2.COLOR_BGR2GRAY) #
convert to grayscale
blur = cv2.blur(gray,
(3, 3)) # blur the
image
ret,
thresh = cv2.threshold(blur, 50, 255,
cv2.THRESH_BINARY)
```

C++

```
Mat gray,
blur_image,
threshold_output;
cvtColor(src, gray,
COLOR_BGR2GRAY); //
convert to
grayscale
blur(gray,
blur_image,
Size(3,
3)); //
apply blur
to
grayscaled
image
threshold(blur_image,
threshold_output, 50,
255,
THRESH_BINARY); //
apply binary
thresholding
```

As you can see, we have converted the image into binary blobs. We next need to find the boundaries of these blobs.

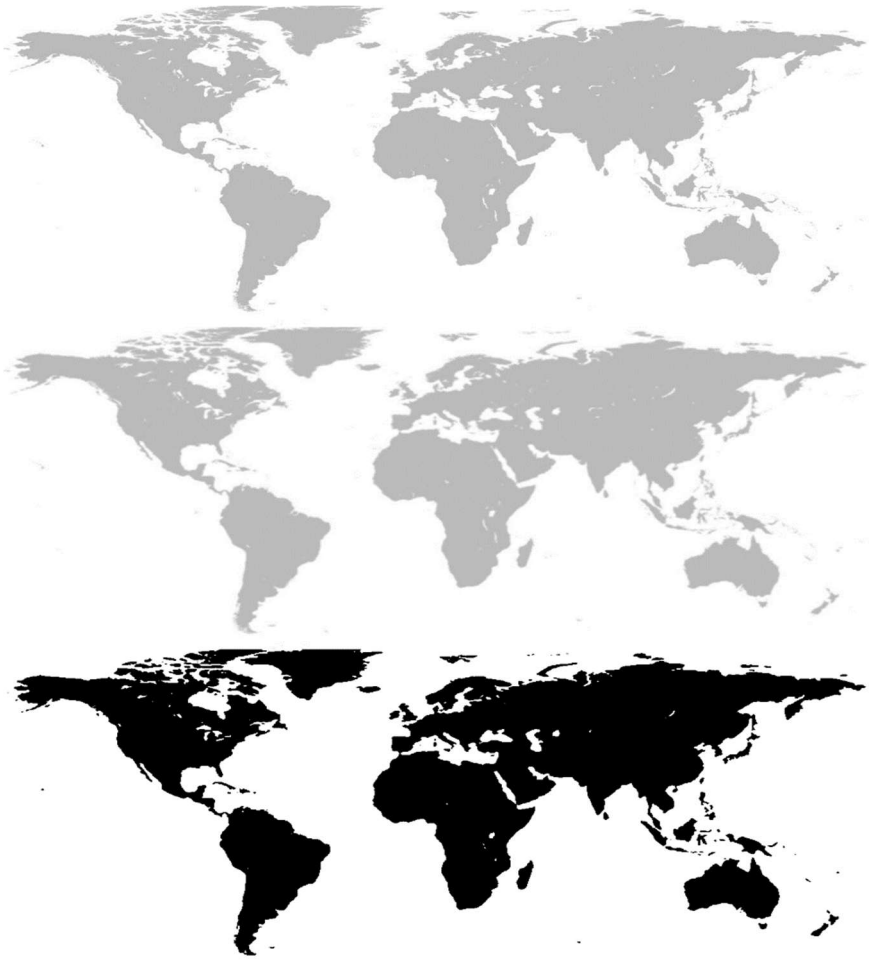


Figure 3: Topmost: Grayscaled Image. Middle: Blurred Image. Bottom: Thresholded Image

- **Step 3: Use findContour to find contours**

Next, we find the contour around every continent using the **findContour** function in OpenCV. Finding the contours gives us a list of boundary points around each blob.

If you are a beginner, you may be tempted to think why we did not simply use edge detection? Edge detection would have simply given us locations of the edges. But we are interested in finding how edges are connected to each other. **findContour** allows us to find those connections and returns the points that form a contour as a list.

Python

```
# Finding
contours
for the
thresholded
image
im2, contours,
hierarchy = cv2.findContours(thresh,
cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

C++

```
vector<
vector<Point>
>
contours; //
list of
contour
points
vector<Vec4i>
hierarchy;
// find
contours
findContours(threshold_output,
contours, hierarchy,
RETR_TREE,
CHAIN_APPROX_SIMPLE, Point(0,
0));
```

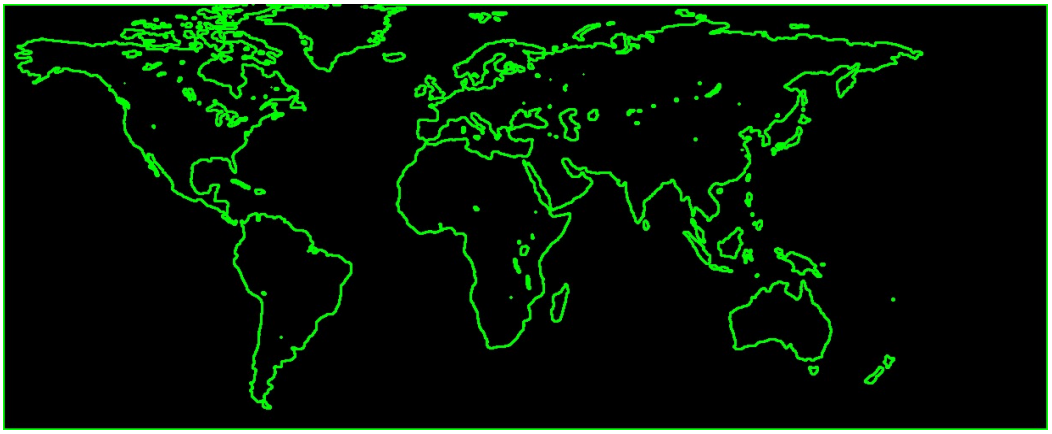


Fig. 4 Output of Finding Contours of the Thresholded Image

- **Step 4: Find the Convex Hull using convexHull**

Since we have found the contours, we can now find the Convex Hull for each of the contours. This can be done using **convexHull** function.

Python

```
#
create
hull
array
for
convex
hull
points
hull = []


#
calculate
points
for each
contour
for i in range(len(contours)):
    #
creating
convex
hull
object
for each
contour
    hull.append(cv2.convexHull(contours[i], False))
```

C++

```
1    //
     create
     hull
     array
     for
     convex
     hull
     points
2    vector< vector<Point>
     >
     hull(contours.size());
3    for(int i = 0; i
     <
     contours.size();
     i++)
4        convexHull(Mat(contours[i]),
     hull[i], False);
```

- **Step 5: Draw the Convex Hull**

The final step is to visualize the convex hulls we have just found. Of course a convex hull itself is just a contour and therefore we can use OpenCV's **drawContours**.

Python

```
1    #
     create
     an
     empty
     black
     image
2    drawing = np.zeros((thresh.shape[0],
     thresh.shape[1], 3), np.uint8)
3
4    # draw
     contours
     and hull
     points
```

```
for i in range(len(contours)):
    color_contours = (0, 255, 0) # green - color for contours
    color = (255, 0, 0) # blue - color for convex hull
    # draw ith contour
    cv2.drawContours(drawing, contours, i, color_contours, 1, 8, hierarchy)
    # draw ith convex hull object
    cv2.drawContours(drawing, hull, i, color, 1, 8)
```

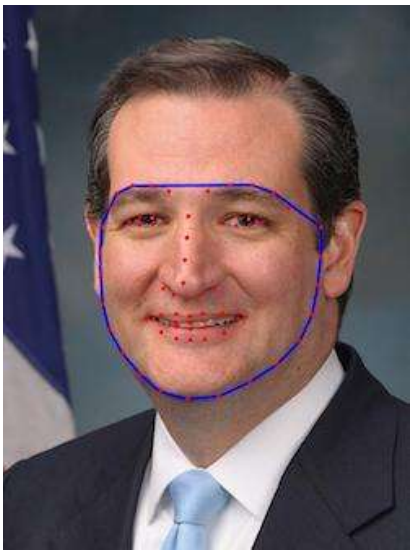
C++

```
// create a blank image (black image)
Mat drawing = Mat::zeros(threshold_output.size(), CV_8UC3);

for(int i = 0; i < contours.size(); i++) [
```

```
5       Scalar color_contours = Scalar(0, 255, 0); // green - color for contours
6       Scalar color = Scalar(255, 0, 0); // blue - color for convex hull
7       // draw ith contour
8       drawContours(drawing, contours, i, color_contours, 1, 8, vector<Vec4i>(), 0, Point());
9       // draw ith convex hull
10      drawContours(drawing, hull, i, color, 1, 8, vector<Vec4i>(), 0, Point());
11  }
```

- **Applications of Convex Hull**
- **Boundary from a set of points**

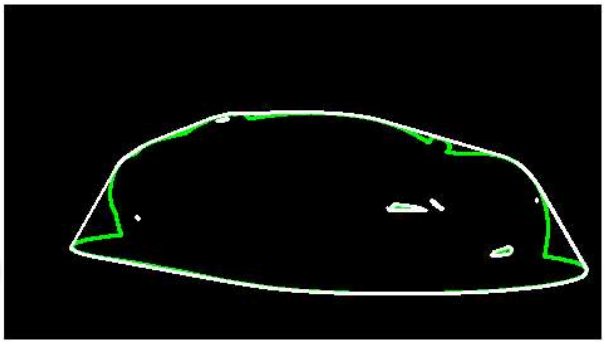
Regular readers of this blog may be aware we have used convexHull before in our face swap application. Given the facial landmarks detected using Dlib, we found the boundary of the face using the convex hull



There are several other applications, where we recover feature points information instead of contour information. For example, in many active lighting systems like Kinect, we recover a grayscale depth map that is a collection of points. We can use the convex hull of these points to locate the boundary of an object in the scene

- **Collision Avoidance**

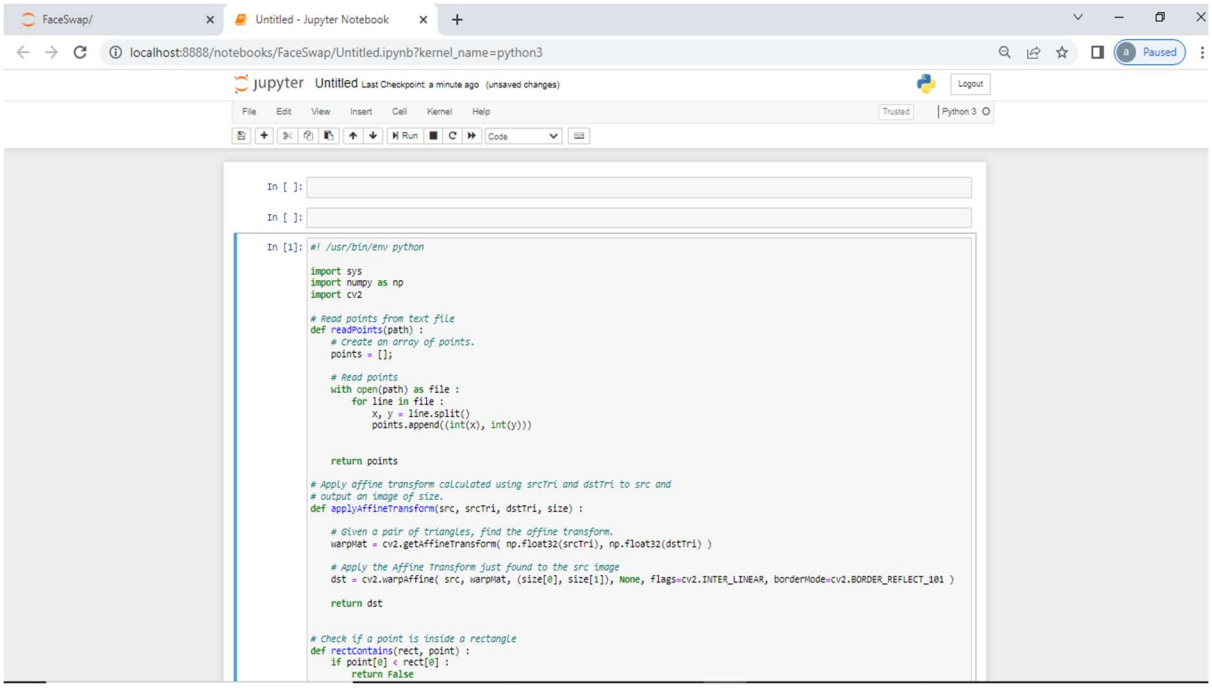
Imagine a car as a set of points and the polygon (minimal set) containing all the points. Now if the convex hull is able to avoid the obstacles, so should the car. Finding the intersection between arbitrary contours is computationally much more expensive than finding the collision between two convex polygons. So you are better off using a convex hull for collision detection or avoidance.



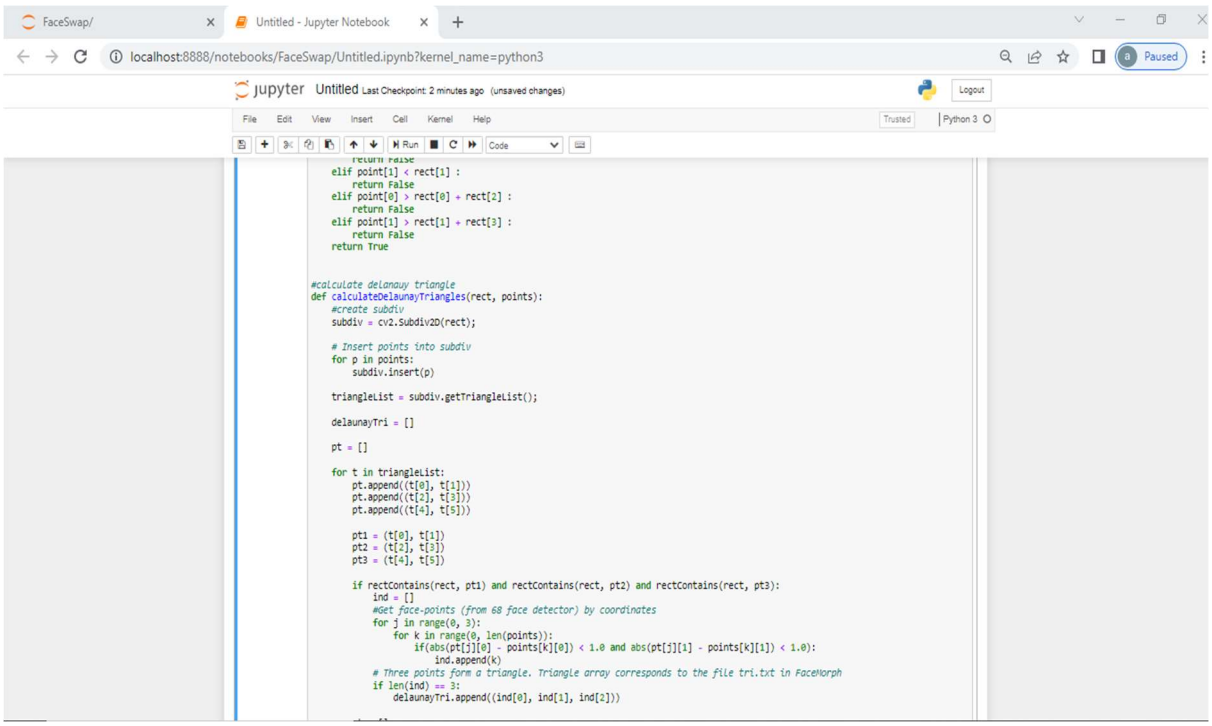
Chapter-3

3.1 Result section:

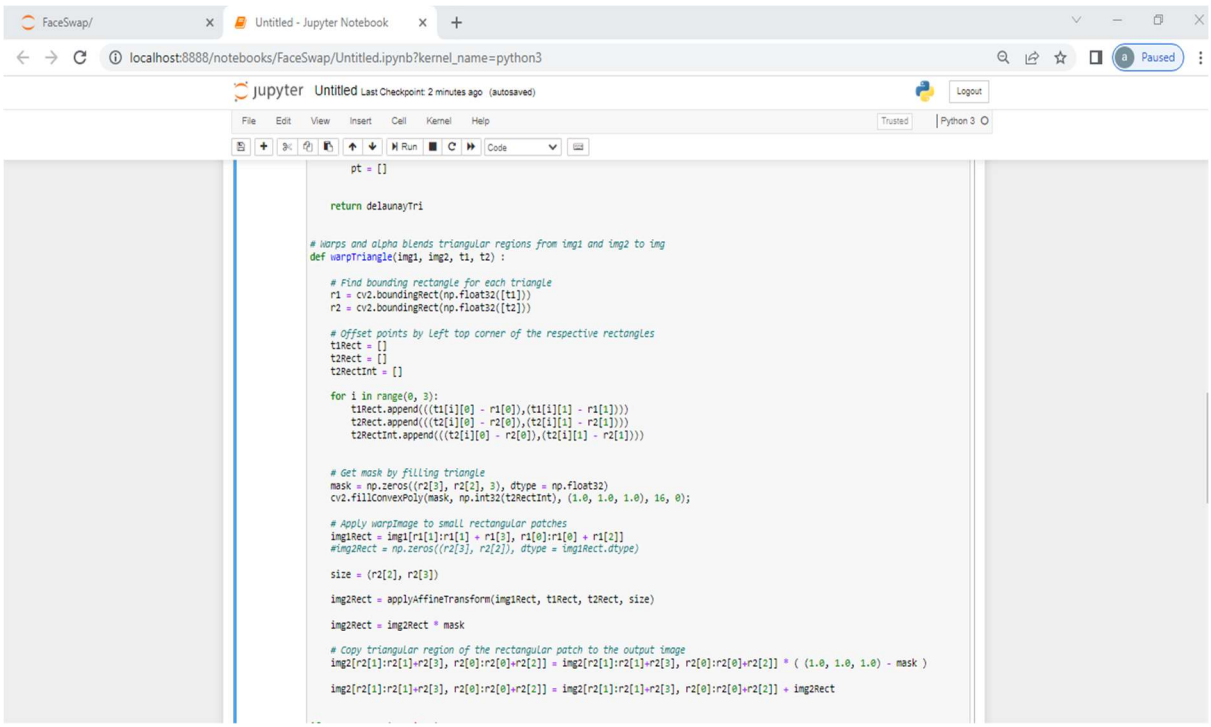
3.1.1 Snapshot of overall program:

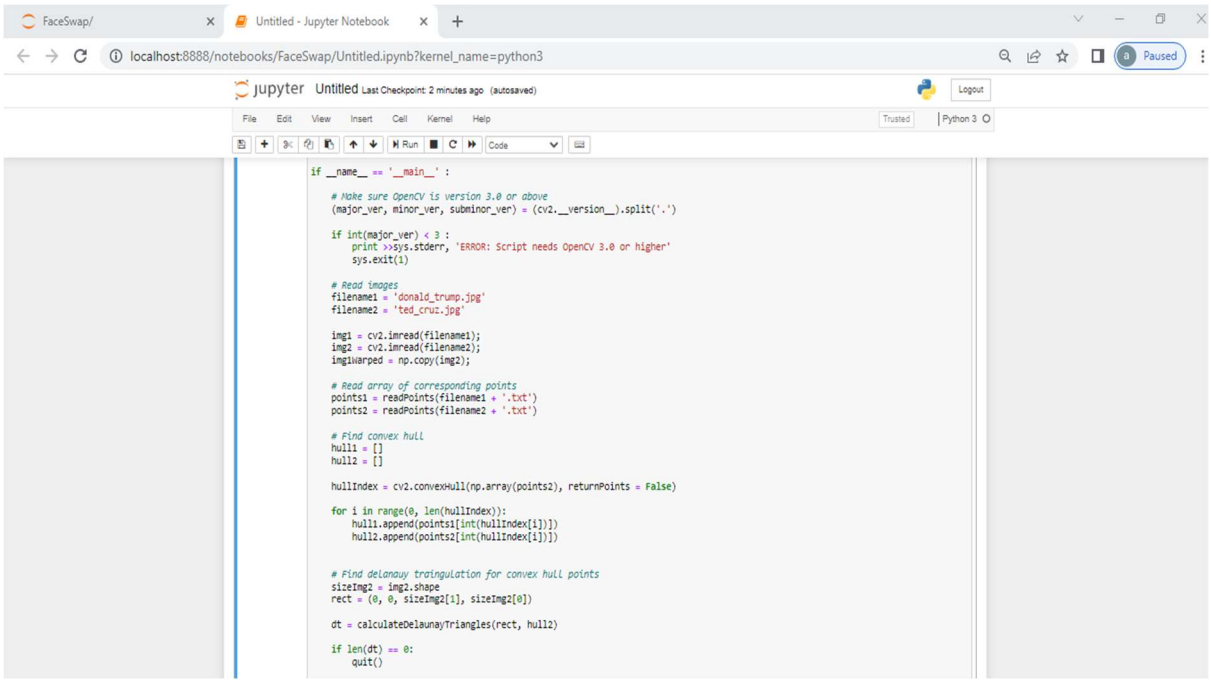


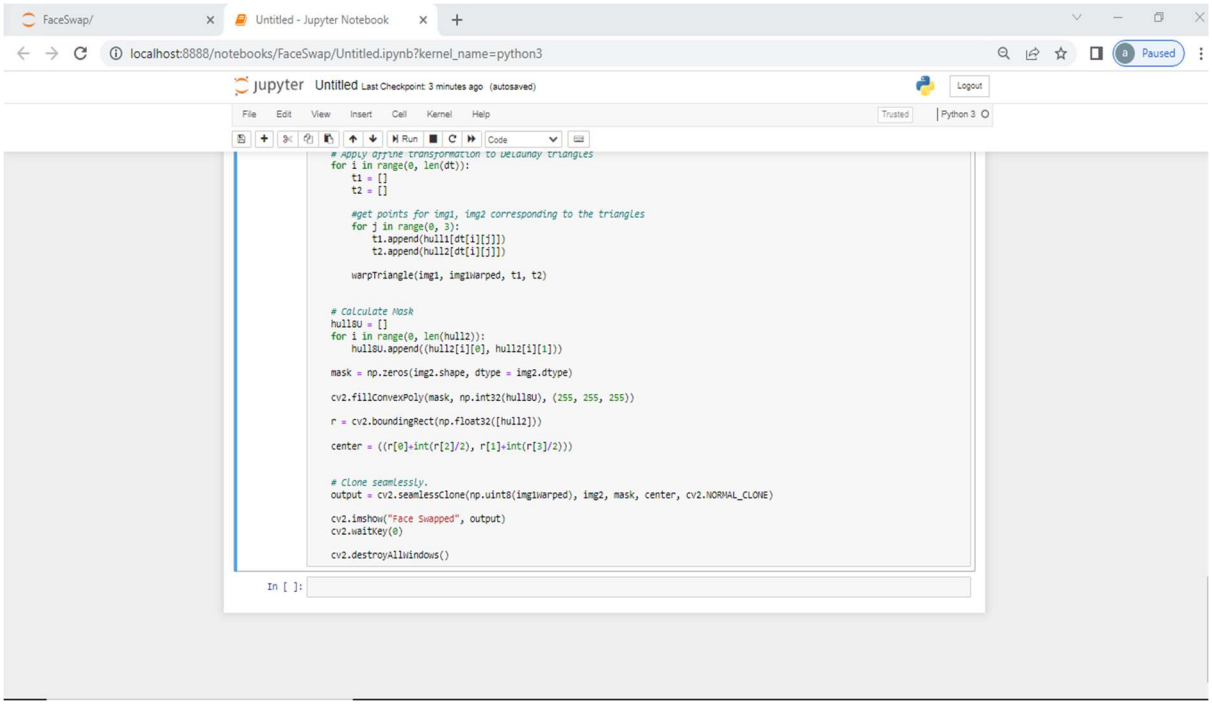
```
In [ ]:

In [ ]:

In [1]: #! /usr/bin/env python

import sys
import numpy as np
import cv2

# Read points from text file
def readPoints(path) :
    # Create an array of points.
    points = [];

    # Read points
    with open(path) as file :
        for line in file :
            x, y = line.split()
            points.append((int(x), int(y)))


    return points

# Apply affine transform calculated using srcTri and dstTri to src and
# output an image of size.
def applyAffineTransform(src, srcTri, dstTri, size) :

    # Given a pair of triangles, find the affine transform.
    warpMat = cv2.getAffineTransform( np.float32(srcTri), np.float32(dstTri) )

    # Apply the Affine Transform just found to the src image
    dst = cv2.warpAffine( src, warpMat, (size[0], size[1]), None, flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT_101 )

    return dst


# Check if a point is inside a rectangle
def rectContains(rect, point) :
    if point[0] < rect[0] :
        return False
```

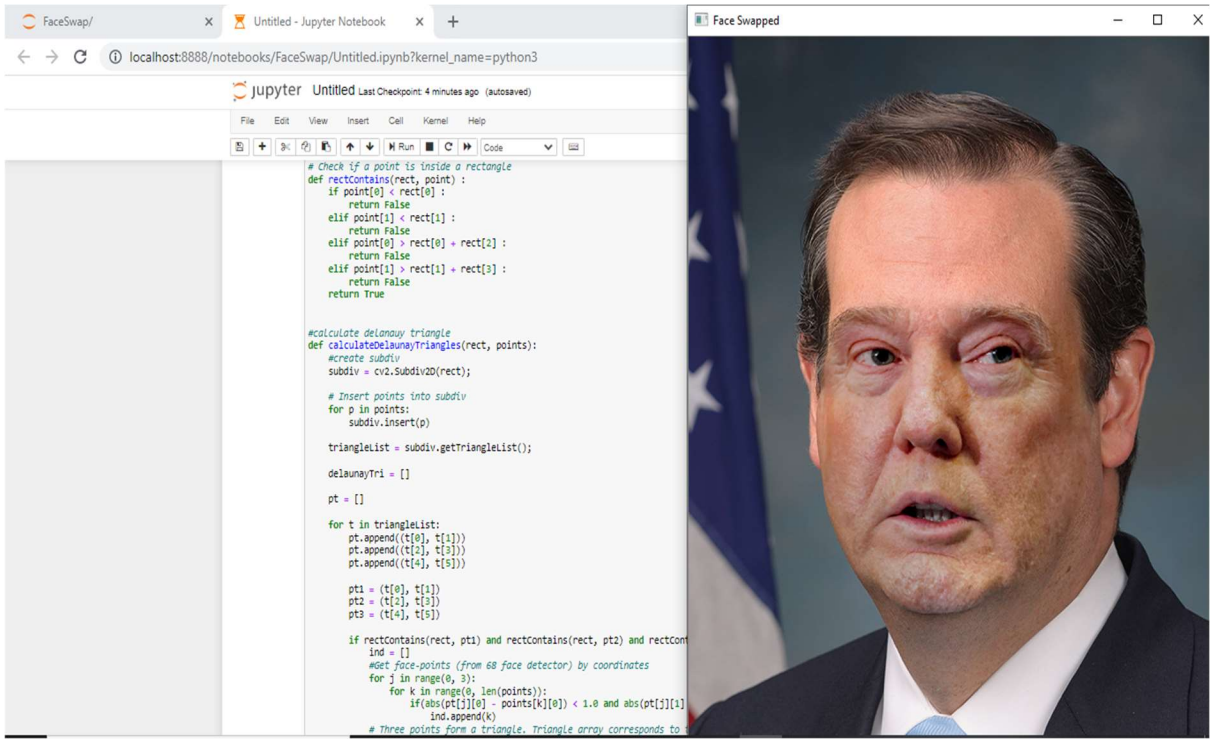


```
        return False
    elif point[1] < rect[1] :
        return False
    elif point[0] > rect[0] + rect[2] :
        return False
    elif point[1] > rect[1] + rect[3] :
        return False
    return True


#calculate delanauy triangle
def calculateDelaunayTriangles(rect, points):
    #create subdiv
    subdiv = cv2.Subdiv2D(rect);

    # Insert points into subdiv
    for p in points:
        subdiv.insert(p)

    triangleList = subdiv.getTriangleList();

    delaunayTri = []

    pt = []

    for t in triangleList:
        pt.append((t[0], t[1]))
        pt.append((t[2], t[3]))
        pt.append((t[4], t[5]))

        pt1 = (t[0], t[1])
        pt2 = (t[2], t[3])
        pt3 = (t[4], t[5])

        if rectContains(rect, pt1) and rectContains(rect, pt2) and rectContains(rect, pt3):
            ind = []
            #Get face-points (from 68 face detector) by coordinates
            for j in range(0, 3):
                for k in range(0, len(points)):
                    if(abs(pt[j][0] - points[k][0]) < 1.0 and abs(pt[j][1] - points[k][1]) < 1.0):
                        ind.append(k)
            # Three points form a triangle. Triangle array corresponds to the file tri.txt in FaceMorph
            if len(ind) == 3:
                delaunayTri.append((ind[0], ind[1], ind[2]))
```


FaceSwap/
Untitled - Jupyter Notebook
localhost:8888/notebooks/FaceSwap/Untitled.ipynb?kernel_name=python3
Paused
jupyter Untitled Last Checkpoint: 2 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code
pt = []
return delaunayTri
Warps and alpha blends triangular regions from img1 and img2 to img
def warpTriangle(img1, img2, t1, t2) :
# Find bounding rectangle for each triangle
r1 = cv2.boundingRect(np.float32([t1]))
r2 = cv2.boundingRect(np.float32([t2]))
# Offset points by left top corner of the respective rectangles
t1Rect = []
t2Rect = []
t2RectInt = []
for i in range(0, 3):
t1Rect.append(((t1[i][0] - r1[0]),(t1[i][1] - r1[1])))
t2Rect.append(((t2[i][0] - r2[0]),(t2[i][1] - r2[1])))
t2RectInt.append(((t2[i][0] - r2[0]),(t2[i][1] - r2[1])))
# Get mask by filling triangle
mask = np.zeros((r2[3], r2[2], 3), dtype = np.float32)
cv2.fillConvexPoly(mask, np.int32(t2RectInt), (1.0, 1.0, 1.0), 16, 0);
# Apply warpImage to small rectangular patches
img1Rect = img1[r1[1]:r1[1] + r1[3], r1[0]:r1[0] + r1[2]]
#img2Rect = np.zeros((r2[3], r2[2]), dtype = img1Rect.dtype)
size = (r2[2], r2[3])
img2Rect = applyAffineTransform(img1Rect, t1Rect, t2Rect, size)
img2Rect = img2Rect * mask
# Copy triangular region of the rectangular patch to the output image
img2[r2[1]:r2[1]+r2[3], r2[0]:r2[0]+r2[2]] = img2[r2[1]:r2[1]+r2[3], r2[0]:r2[0]+r2[2]] * ((1.0, 1.0, 1.0) - mask)
img2[r2[1]:r2[1]+r2[3], r2[0]:r2[0]+r2[2]] = img2[r2[1]:r2[1]+r2[3], r2[0]:r2[0]+r2[2]] + img2Rect


FaceSwap/
Untitled - Jupyter Notebook
localhost:8888/notebooks/FaceSwap/Untitled.ipynb?kernel_name=python3
Paused
jupyter Untitled Last Checkpoint: 2 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code
if __name__ == '__main__' :
Make sure OpenCV is version 3.0 or above
(major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')
if int(major_ver) < 3 :
print >>sys.stderr, 'ERROR: Script needs OpenCV 3.0 or higher'
sys.exit(1)
# Read images
filename1 = 'donald_trump.jpg'
filename2 = 'ted_cruz.jpg'
img1 = cv2.imread(filename1);
img2 = cv2.imread(filename2);
img1Warped = np.copy(img2);
# Read array of corresponding points
points1 = readPoints(filename1 + '.txt')
points2 = readPoints(filename2 + '.txt')
# Find convex hull
hull1 = []
hull2 = []
hullIndex = cv2.convexHull(np.array(points2), returnPoints = False)
for i in range(0, len(hullIndex)):
hull1.append(points1[int(hullIndex[i])])
hull2.append(points2[int(hullIndex[i])])
# Find delanauy traingulation for convex hull points
sizeImg2 = img2.shape
rect = (0, 0, sizeImg2[1], sizeImg2[0])
dt = calculateDelaunayTriangles(rect, hull2)
if len(dt) == 0:
quit()

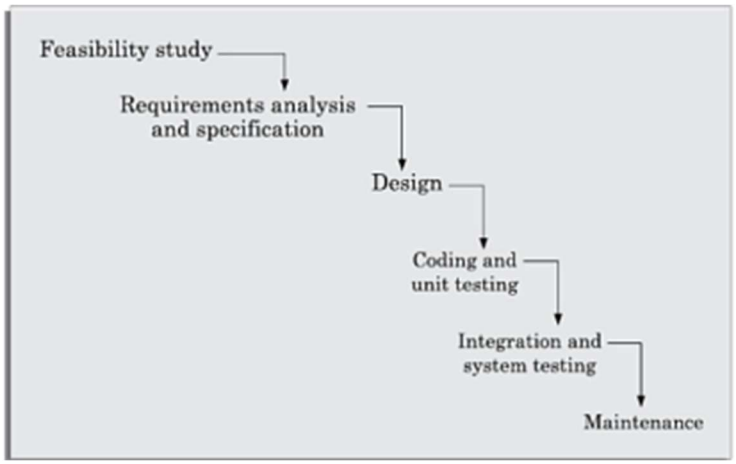

FaceSwap/
Untitled - Jupyter Notebook
localhost:8888/notebooks/FaceSwap/Untitled.ipynb?kernel_name=python3
jupyter Untitled Last Checkpoint: 3 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code
# Apply affine transformation to Delaunay triangles
for i in range(0, len(dt)):
t1 = []
t2 = []
#get points for img1, img2 corresponding to the triangles
for j in range(0, 3):
t1.append(hull1[dt[i][j]])
t2.append(hull2[dt[i][j]])
warpTriangle(img1, img1Warped, t1, t2)
# Calculate Mask
hull8U = []
for i in range(0, len(hull2)):
hull8U.append((hull2[i][0], hull2[i][1]))
mask = np.zeros(img2.shape, dtype = img2.dtype)
cv2.fillConvexPoly(mask, np.int32(hull8U), (255, 255, 255))
r = cv2.boundingRect(np.float32([hull2]))
center = ((r[0]+int(r[2]/2), r[1]+int(r[3]/2)))
# Clone seamlessly.
output = cv2.seamlessClone(np.uint8(img1Warped), img2, mask, center, cv2.NORMAL_CLONE)
cv2.imshow("Face Swapped", output)
cv2.waitKey(0)
cv2.destroyAllWindows()
In [ ]:


FaceSwap/
Untitled - Jupyter Notebook
localhost:8888/notebooks/FaceSwap/Untitled.ipynb?kernel_name=python3
jupyter Untitled Last Checkpoint: 4 minutes ago (autosaved)
File Edit View Insert Cell Kernel Help
Run
Code
Check if a point is inside a rectangle
def rectContains(rect, point) :
if point[0] < rect[0] :
return False
elif point[1] < rect[1] :
return False
elif point[0] > rect[0] + rect[2] :
return False
elif point[1] > rect[1] + rect[3] :
return False
return True
#calculate delanauy triangle
def calculateDelaunayTriangles(rect, points):
#create subdiv
subdiv = cv2.Subdiv2D(rect);
# Insert points into subdiv
for p in points:
subdiv.insert(p)
triangleList = subdiv.getTriangleList();
delaunayTri = []
pt = []
for t in triangleList:
pt.append((t[0], t[1]))
pt.append((t[2], t[3]))
pt.append((t[4], t[5]))
pt1 = (t[0], t[1])
pt2 = (t[2], t[3])
pt3 = (t[4], t[5])
if rectContains(rect, pt1) and rectContains(rect, pt2) and rectCont
ind = []
#Get face-points (from 68 face detector) by coordinates
for j in range(0, 3):
for k in range(0, len(points)):
if(abs(pt[j][0] - points[k][0]) < 1.0 and abs(pt[j][1]
ind.append(k)
# Three points form a triangle. Triangle array corresponds to
Face Swapped

3.1.2 Software Development :

Classical waterfall model is intuitively the most obvious way to develop software. It is simple but idealistic. In fact, it is hard to put this model into use in any non-trivial software development project

The classical waterfall model divides the life cycle into a set of phases as shown in Figure



- **Feasibility study**

feasibility study involves carrying out several activities such as collection of basic information relating to the software such as the different data items that would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system, as well as various constraints on the development.

- **Requirements analysis and specification**

Requirements gathering and analysis:

The goal of the requirements gathering activity is to collect all relevant information regarding the software to be developed from the customer with a view to clearly understand the requirements. For this, first requirements are gathered from the customer and then the gathered requirements are
analysed. The goal of the requirements analysis activity is to weed out the incompleteness and inconsistencies in these gathered requirements. Note that an inconsistent requirement is one in which some part of the requirement contradicts with some other part. On the other hand, a n incomplete requirement is one in which some parts of the actual requirements have been omitted.

Requirements specification:

After the requirement gathering and analysis activities are complete, the identified requirements are documented. This is called a software requirements specification (SRS) document. The SRS document is written using end-user terminology. This makes the SRS document understandable to the customer. Therefore, understandability of the SRS document is an important issue. The SRS document normally serves as a contract between the development team and the customer. Any future dispute between the customer and the developers can be settled by examining the SRS document. The SRS document is therefore an important document which must be thoroughly understood by the development team, and reviewed jointly with the customer. The SRS document not only forms the basis for carrying out all the development activities, but several documents such as users' manuals, system test plan, etc.

❖ Design

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the SRS document. Two distinctly different design approaches are popularly being used at present—the procedural and object-oriented design approaches. In the following, we briefly discuss the essence of these two approaches. The traditional design approach is in use in many software development projects at the present time. This traditional design technique is based on the

data flow-oriented design approach. It consists of two important activities; first structured analysis of the requirements specification is carried out where the detailed structure of the problem is examined. This is followed by a structured design step where the results of structured analysis are transformed into the software design. During structured analysis, the functional requirements specified in the SRS document are decomposed into subfunctions and the data-flow among these subfunctions is analysed and represented diagrammatically in the form of DFDs. The DFD technique is discussed in Chapter 6. Structured design is undertaken once the structured analysis activity is complete. Structured design consists of two main activities—architectural design (also called high-level design) and detailed design (also called Low-level design). High-level design involves decomposing the system into modules, and representing the interfaces and the invocation relationships among the modules. A high-level software design is some times referred to as the software architecture. During the detailed design activity, internals of the individual modules such as the data structures and algorithms of the modules are designed and documented. Object-oriented design approach: In this technique, various objects that occur in the problem domain and the solution domain are first identified and the different relationships that exist among these objects are identified. The object structure is further refined to obtain the detailed design. The OOD approach is credited to have several benefits such as lower development time and effort, and better maintainability of the software.

❖ Coding and unit testing

The purpose of the coding and unit testing phase is to translate a software design into source code and to ensure that individually each function is working correctly. The coding phase is also sometimes called the implementation phase, since the design is implemented into a workable solution in this phase. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually unit tested. The

main objective of unit testing is to determine the correct working of the individual modules. The specific activities carried out during unit testing include designing test cases, testing, debugging to fix problems, and management of test cases. We shall discuss the coding and unit testing techniques.

Integration and system testing

Integration of different modules is undertaken soon after they have been coded and unit tested. During the integration and system testing phase, the different modules are integrated in a planned manner. Various modules making up a software are almost never integrated in one shot (can you guess the reason for this?). Integration of various modules are normally carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained. System testing is carried out on this fully working system.

System testing usually consists of three different kinds of testing activities:

- **α-testing:** testing is the system testing performed by the development team.
- **β-testing:** This is the system testing performed by a friendly set of customers.
- **Acceptance testing:** After the software has been delivered, the customer performs system testing to determine whether to accept the delivered software or to reject it.

❖ Maintenance

The total effort spent on maintenance of a typical software during its operation phase is much more than that required for developing the software itself. Many studies carried out in the past confirm this and indicate that the ratio of relative effort of developing a typical software product and the total effort spent on its maintenance is roughly 40:60. Maintenance is required in the following three types of situations: Corrective maintenance: This type of

maintenance is carried out to correct errors that were not discovered during the product development phase.

- **Perfective maintenance:** This type of maintenance is carried out to improve the performance of the system, or to enhance the functionalities of the system based on customer's requests.

- **Adaptive maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system.

3.3 Conclusion & Future scope:

Images define the world, each image has its own story, it contains a lot of crucial information that can be useful in many ways. This information can be obtained with the help of the technique known as Image Processing.
It is the core part of computer vision which plays a crucial role in many real-world examples like robotics, self-driving cars, and object detection. Image processing allows us to transform and manipulate thousands of images at a time and extract useful insights from them. It has a wide range of applications in almost every field. Python is one of the widely used programming languages for this purpose. Its amazing libraries and tools help in achieving the task of image processing very efficiently.
Through this article, you will learn about classical algorithms, techniques, and tools to process the image and get the desired output.

There are many use cases in this technology, few of this mention below.,

- Now a days social media is a huge demanding site, not a single user can't take patience for connection error even a small breaches. And also the reels and youtube shorts involve youth a large scale. This reels or face filters also use this type of immersive technology.

- With a hand of good things bads mindset also comes. This technology use in bad things too, use this technology some terrorist organisations and also the against political parties make fake image and news to crate public violence and political violence.

❖ REFERANCE & BOOKS

- https://www.edureka.co/blog/python-opencv-tutorial/
- https://learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/
- https://learnopencv.com/facial-landmark-detection/
- https://learnopencv.com/tag/face-recognition/
- https://learnopencv.com/warp-one-triangle-to-another-using-opencv-c-python/
- https://learnopencv.com/face-swap-using-opencv-c-python/
- https://www.analyticsvidhya.com/blog/2021/07/give-hermione-granger-a-cool-pair-of-glasses-by-building-snapchat-filter-using-opencv/
- https://1lib.in/book/3647627/667c74
- https://1lib.in/book/3306705/d9c0e3?dsource=recommend

❖ Project Contributors

Dr. Partho Choudhuri

Project Mentor

Aditya Das

Project coder

Suparno Sarkar

Project Debuger

Ananya Ghosh

Project Content Writer

Moumita Maity

Project Code tester

To download the softcopy scan the QR code.