RBE/CS549 - Computer Vision
Project 2 - FaceSwap

Anagha Dangle - Mihir Kulkarni
ardangle@wpi.edu - mmkulkarni@wpi.edu
Using 1 late day

***Abstract*—The main task of this project is to perform face swapping which involves replacing the face in a video with a different face from an image and swapping the 2 faces in the same video. This Project is divided into 2 phases. Phase 1 explores face swapping using the traditional approach. Here, we detect the facial landmarks using the *dlib* library, perform face warping, face replacement, and finally blending. Phase 2 involves using an off-the-shelf deep learning model to detect the facial features and perform face replacement in the same way as Phase 1.**

I. PHASE-I: TRADITIONAL APPROACH

This approach is divided into the following 4 steps:

- Facial Landmark detection of both images using *dlib* library.
- Face Warping using Delaunay Triangulation or Thin Plate Spline (TPS). We will see both these approaches in detail later.
- Creating a mask of the destination face.
- Replacing the destination face with the source face.

Let's have a look at these steps in detail.

A. Facial Landmark Detection

The facial features or landmarks of a face are detected using the *dlib* library. We use the get_frontal_face_detector() function for the detector and the shape_predictor() function for the predictor. The detector outputs the bounding box of the face in the image, while the predictor gives the **68 points (coordinates in the image)** that correspond to the specific facial features. The pre-trained facial landmark detector (.dat file) is used to predict these 68 points. An advantage of using *dlib* is that it can detect multiple faces in the image. The detector will return as many rectangular bounding boxes as the number of faces in the image. This is particularly important when you want to swap 2 faces in the same image.

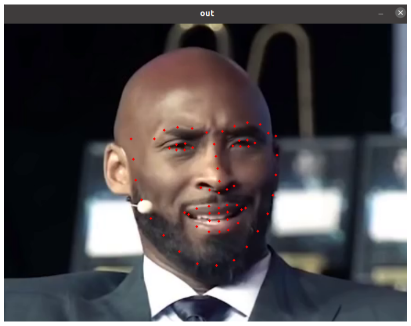

Fig. 1: Features of Kobe

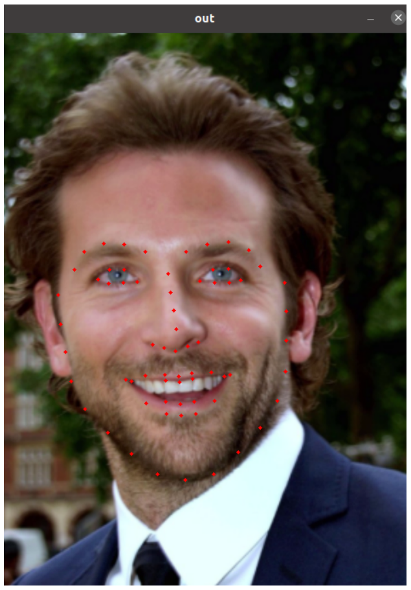

Fig. 2: Facial features of Bradley Cooper

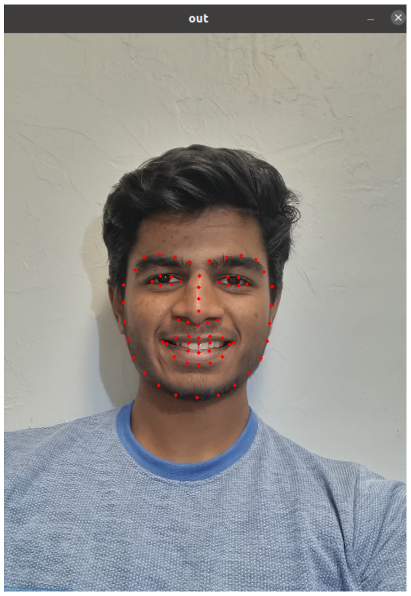

Fig. 3: Facial features of Mihir

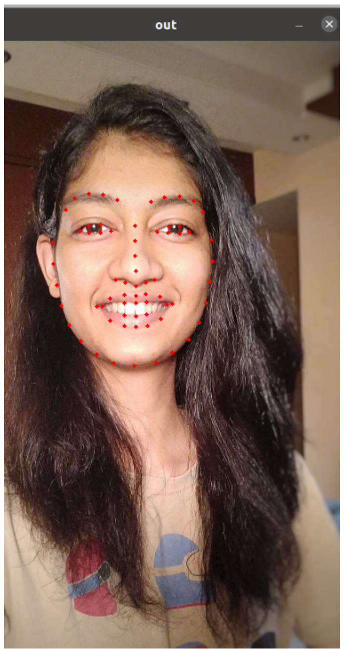

Fig. 4: Facial features of Anagha

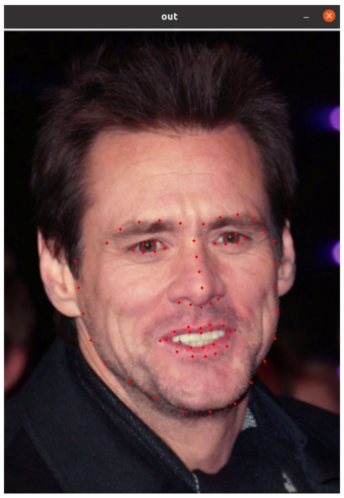

Fig. 5: Facial features of Jim

B. Face Warping

The 2 approaches here are as follows:

1) Delaunay Triangulation: Now that we have the 68 feature points and the bounding rectangle, we perform the Delaunay triangulation of the image. Using the OpenCV function Subdiv2D(), we get the Delaunay triangles in the form of **x and y coordinates of the vertices of each triangle**. We perform this triangulation for both, the source image and the destination image giving us the vertices coordinates of all triangles in both images. We only select the triangles that belong to the face markers. The triangles correspond because the indices are taken to be the same from both images. Due to this the number of triangles always remains the same. We also tried using bounding boxes to keep the number of triangles the same however it did not seem to work properly.

The next task is to warp the source image according to the shape of the destination image so that the final image doesn't look out of place. For this, we first take the bounding rectangle of each triangle in the image. Then we calculate the shift between the x and y coordinates of the triangle with the x and y coordinates of its corresponding bounding rectangle. Using this shift we affine transform every triangle in the source image to get the warped image.

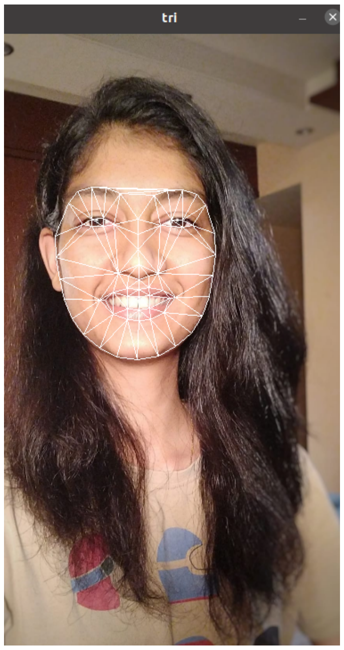

Fig. 6: Delaunay Triangulation of Anagha

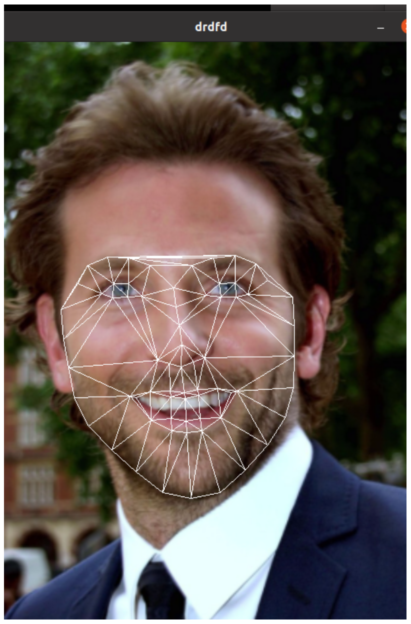

Fig. 7: Delaunay Triangulation of Bradley

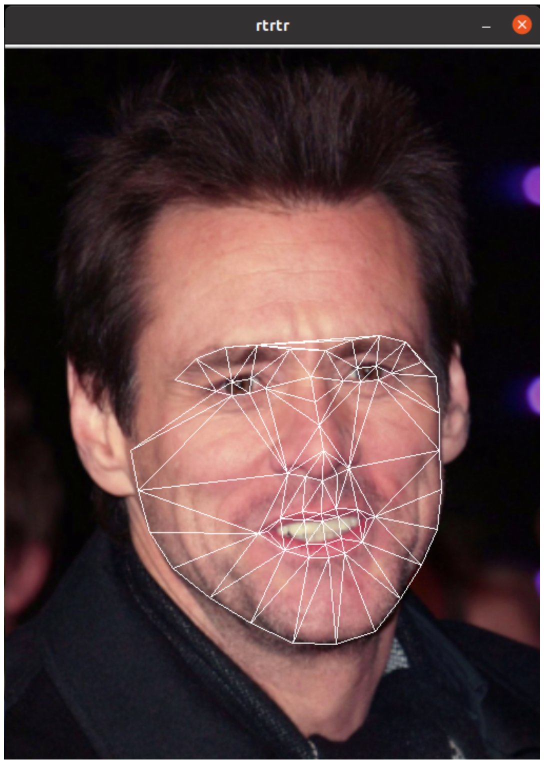

Fig. 8: Delaunay Triangulation of Jim

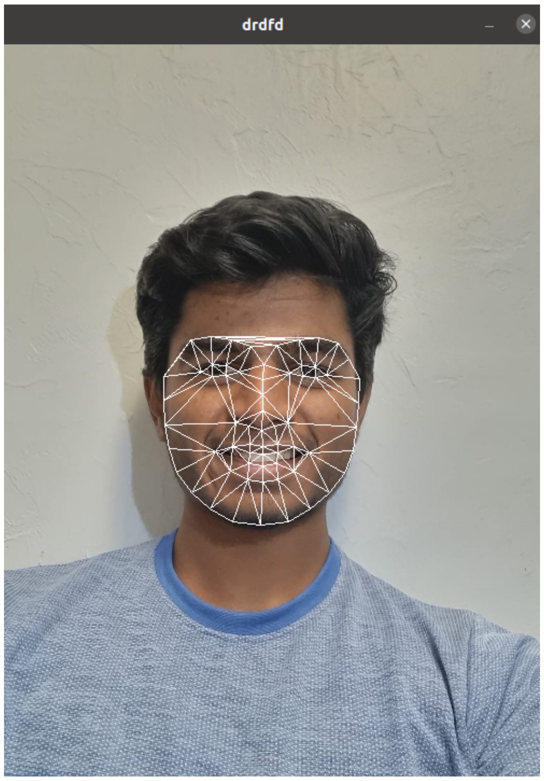

Fig. 9: Delaunay Triangulation of Mihir

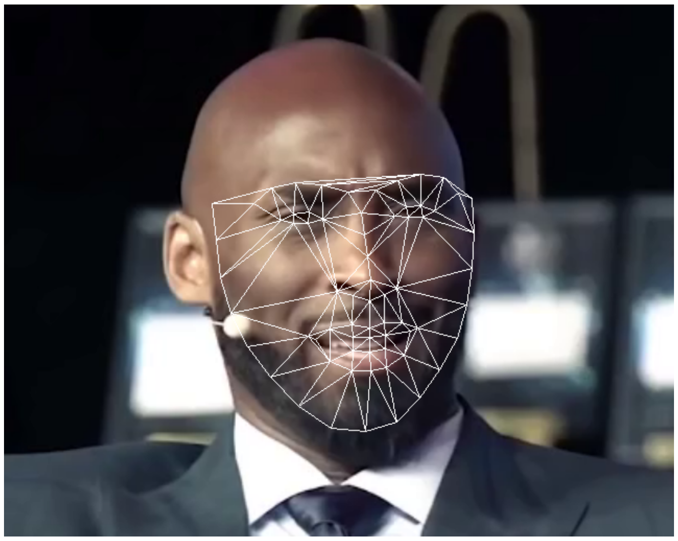

Fig. 10: Delaunay Triangulation of Kobe

2) Thin Plate Spline (TPS) Warping: Thin Plate Spline can model arbitrarily complex shapes like our faces. So instead of using affine transformation on individual triangles, like in Delaunay Triangulation, we use TPS to transform the entire face in a better and uniform manner. We first calculate the L matrix. For that, we need to calculate K, P, and P^T. For this, we need the control and source points. We get a 68x68 matrix of K which is calculated using R and euclidean distance is used for the measure. We use the map coordinates function to interpolate the source points with the calculated TPS output. We have linked a few resources which helped us understand TPS better at the bottom of the page.

C. Mask creation

Now that we have the warped source image, we have to create a mask of the face in the destination image which we want to replace. With the OpenCV function convexHull() and fillConvexPoly(), we generate a mask of the face to be replaced, using the facial landmarks of the destination face. The convexHull function gives an outline of a face in an image while the fillConvex Poly function fills the area bounded by the outline provided by the convex hull function by color of our choice.

D. Replacing/Swapping faces and Blending

We have the warped source image and the mask of the destination face. Now in order to replace the destination face, first we multiply the warped source image with the mask so as to place the source image at the exact desired location of the destination face. Then, we use the OpenCV function seamlessClone() which combines the warped image and the destination image to produce a single blended image. We also use feather blending to have a better output image. We observed the output to be the same and not much different in both cases.

E. Original Images

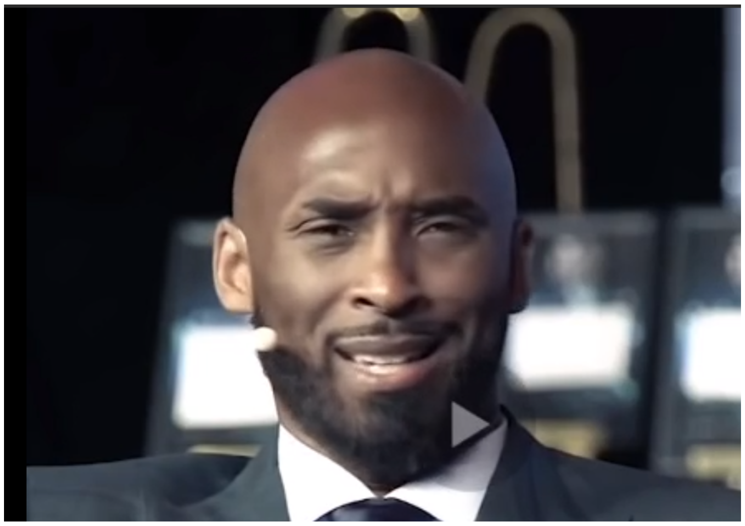

Fig. 11: Original image of Kobe

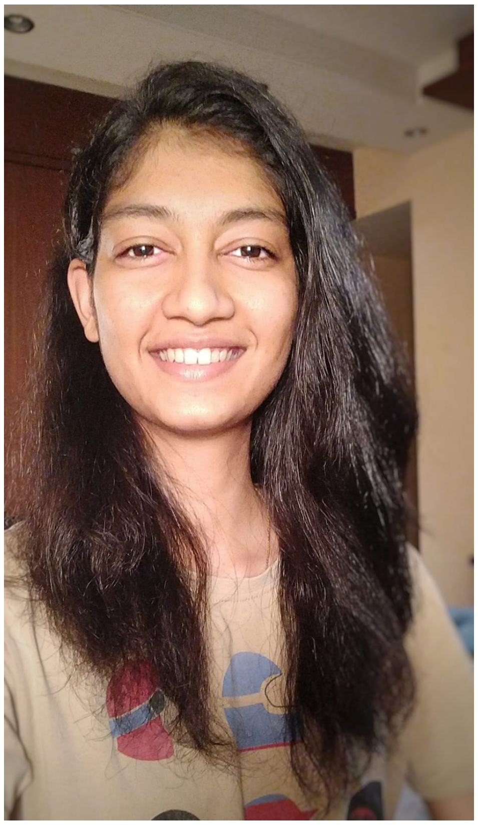

Fig. 12: Original image of Anagha

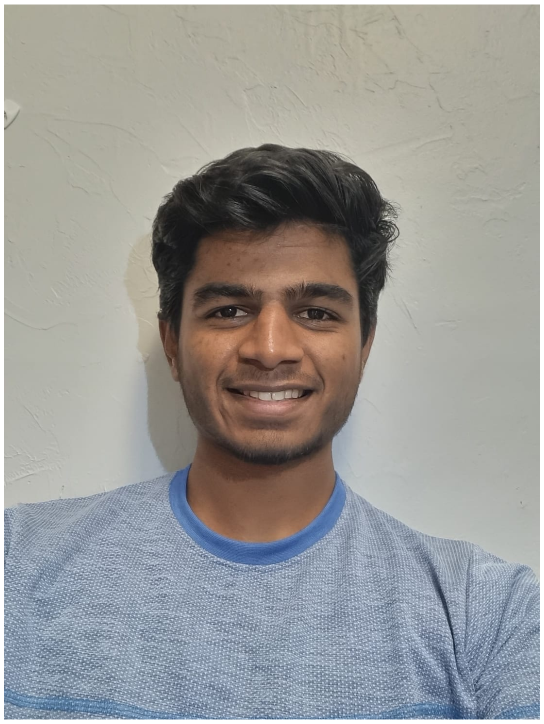

Fig. 13: Original image of Mihir

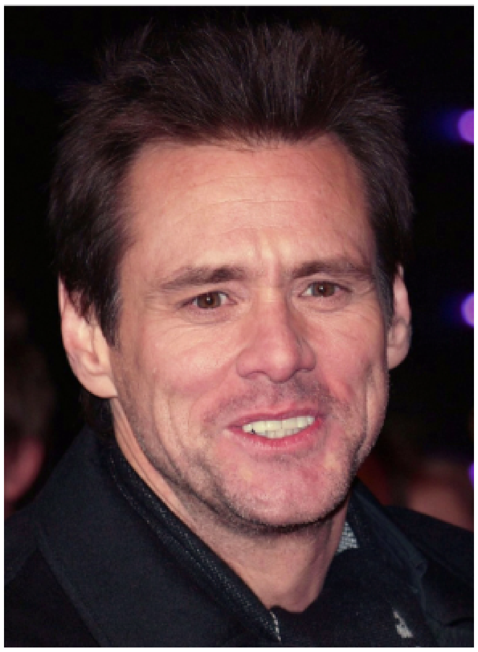

Fig. 14: Original image of Jim

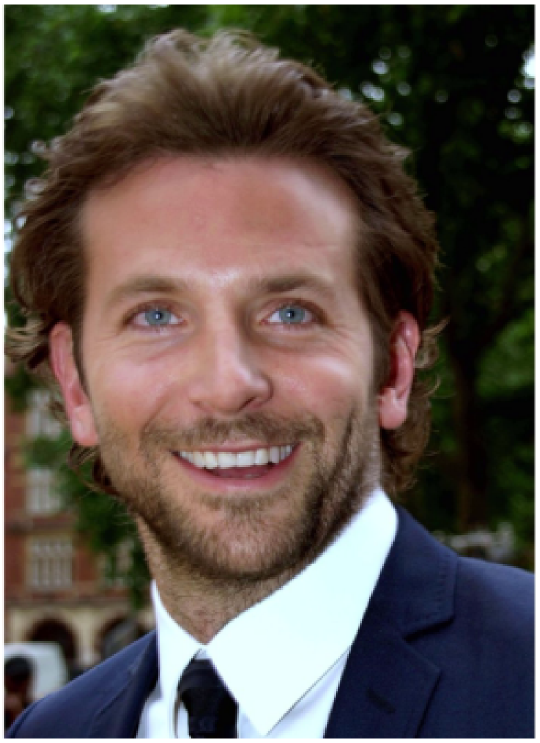
Fig. 15: Original image of Bradley

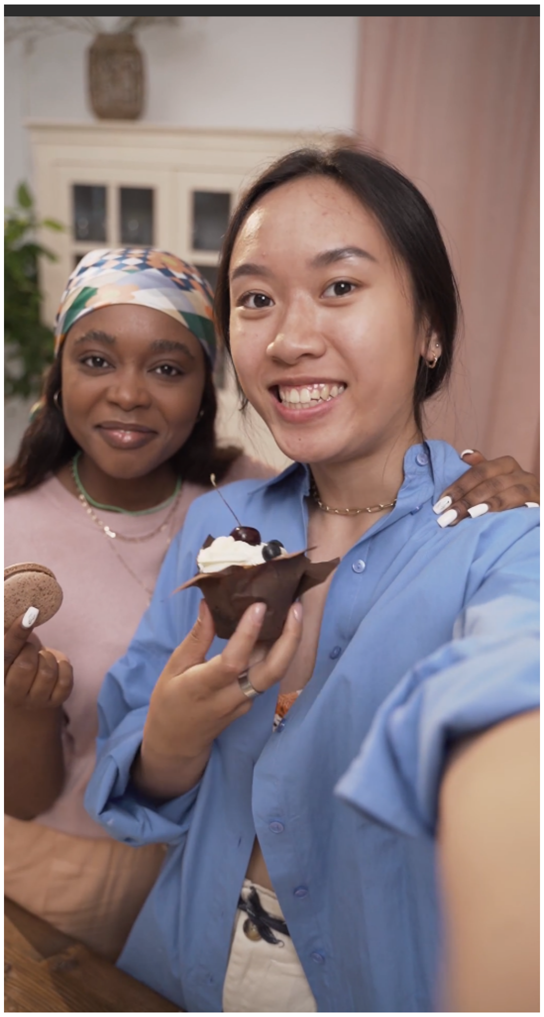
Fig. 17: Original random image

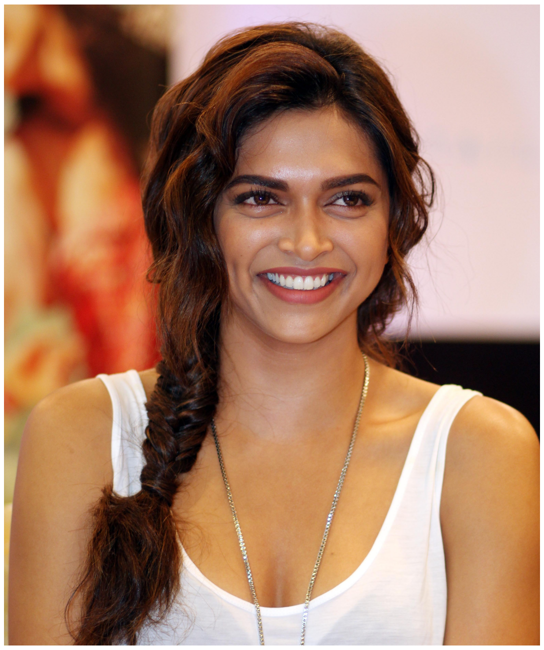
Fig. 16: Original image of Deepika

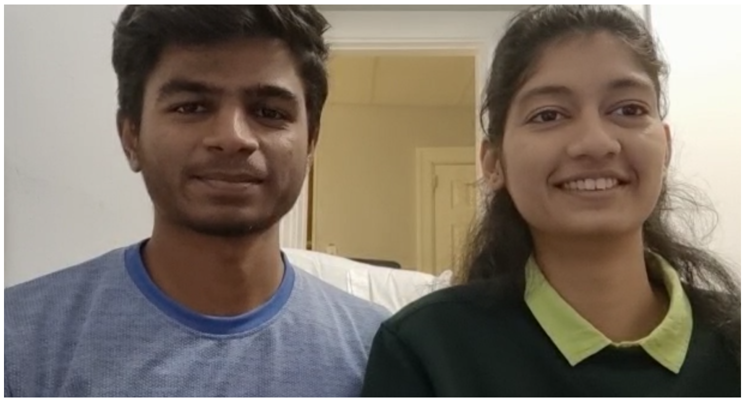
Fig. 18: Original image of Mihir and Anagha

F. Results - Replacement

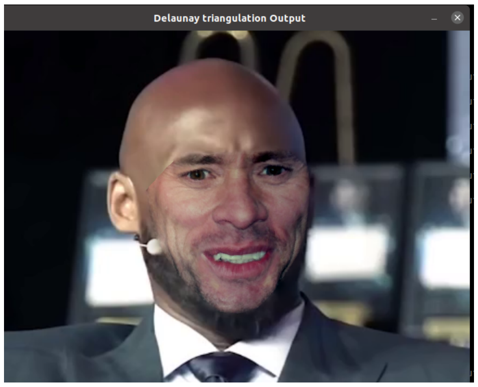

Fig. 19: Delaunay Output of Kobe Jim

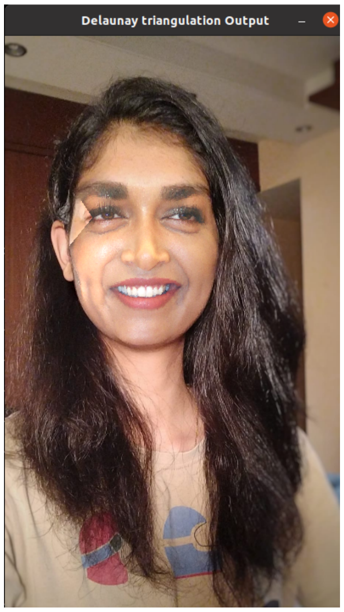

Fig. 20: Delaunay Output of Anagha & Deepika

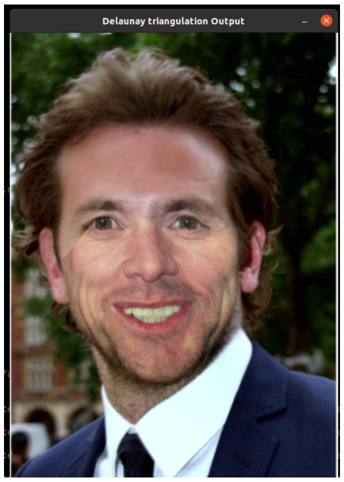

Fig. 21: Delaunay Output of Jim Bradley

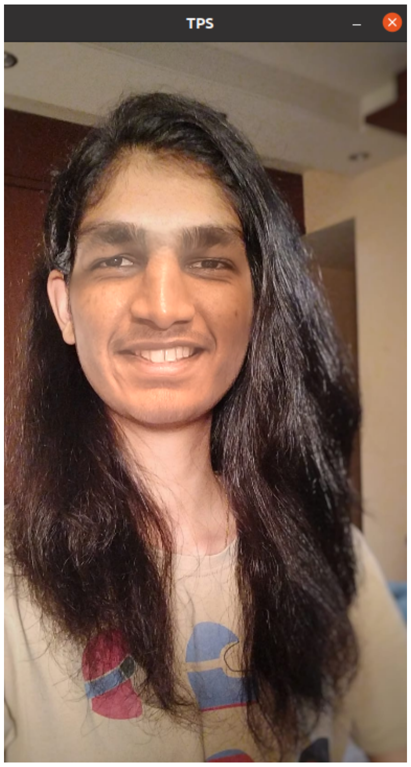

Fig. 22: TPS output of Anagha and Mihir

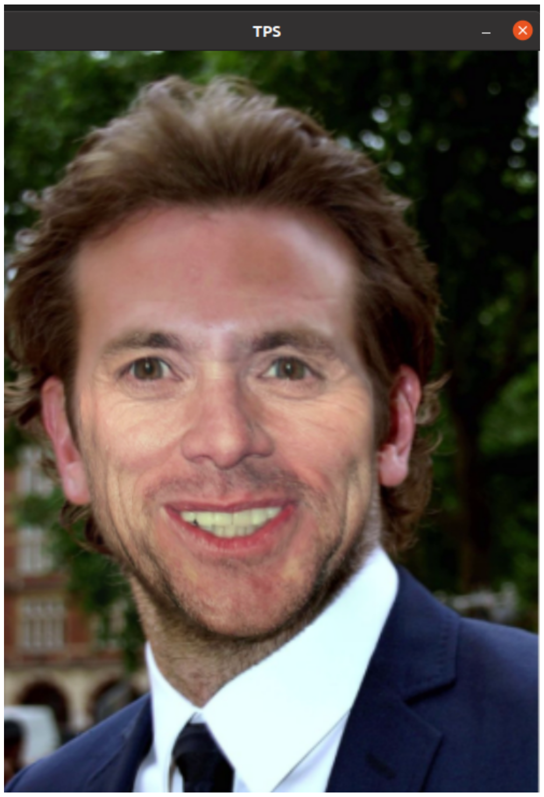

Fig. 23: TPS output of Jim Bradley

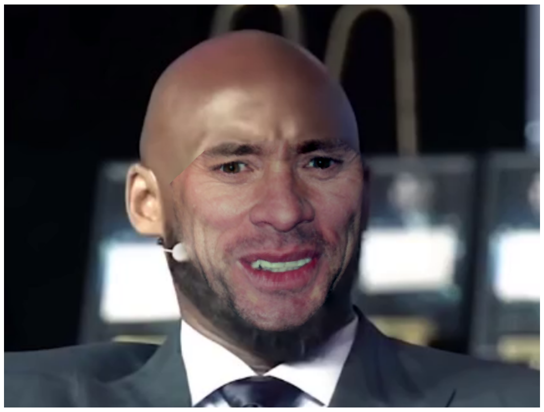
Fig. 24: TPS output of Kobe Jim

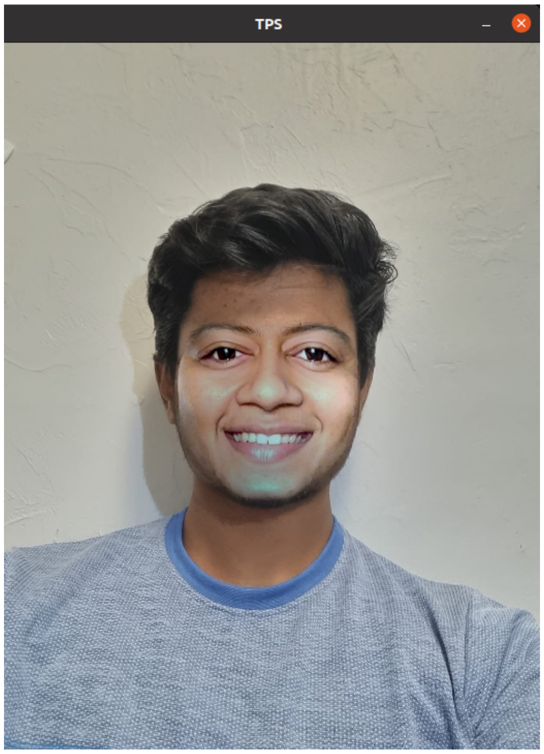

Fig. 25: TPS output of Mihir Anagha

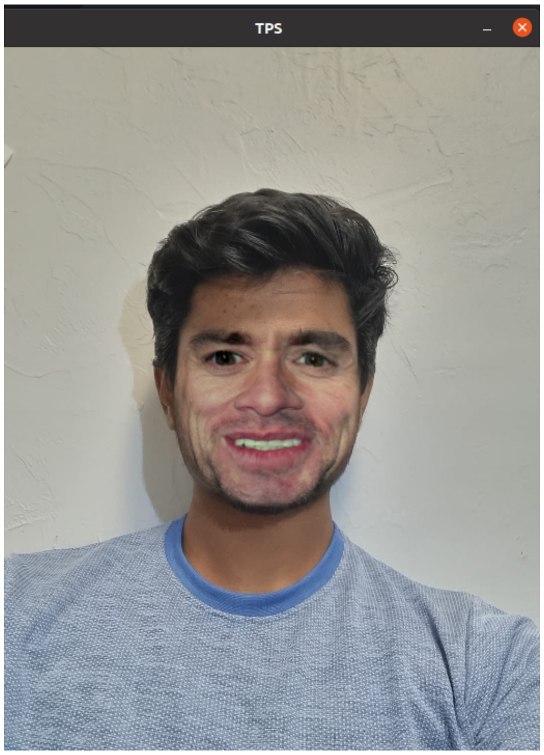

Fig. 26: TPS output of Mihir Jim

G. Results - Swapping

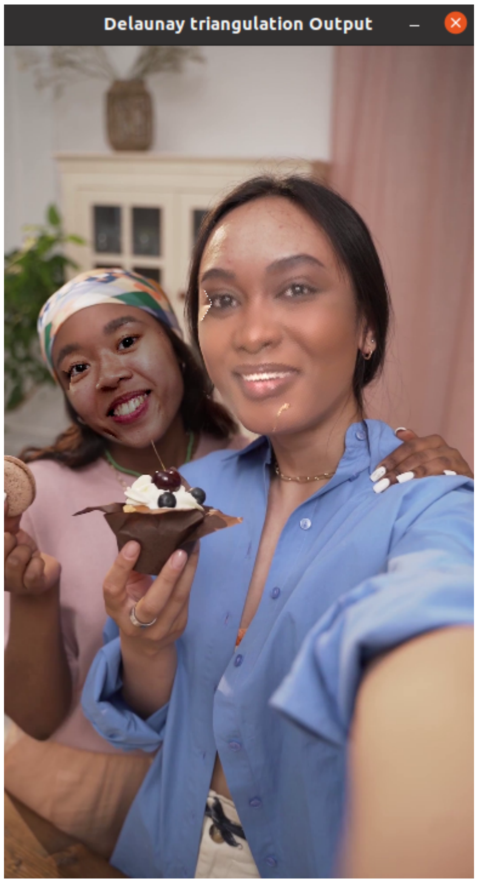

Fig. 27: Delaunay Output of random image

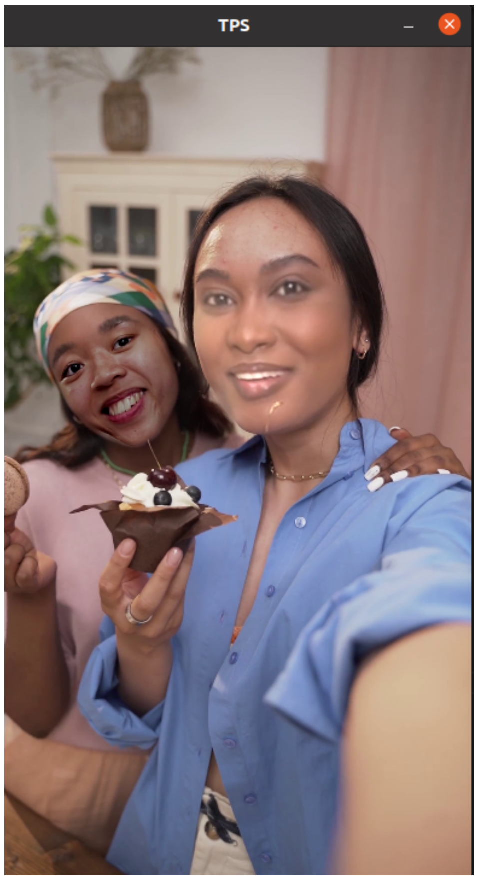

Fig. 29: TPS output of a random video

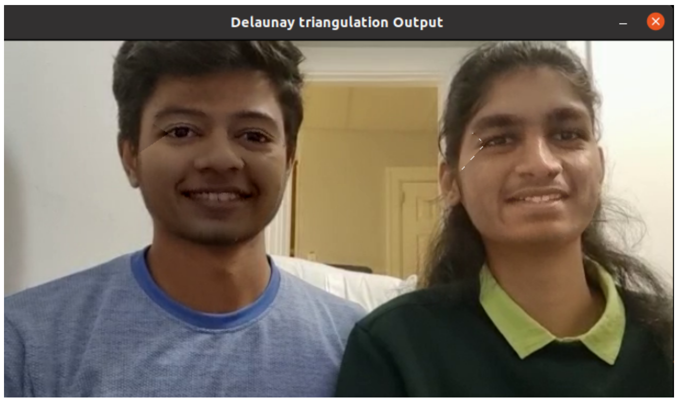

Fig. 28: Face swap using Delaunay Triangulation of Anagha Mihir

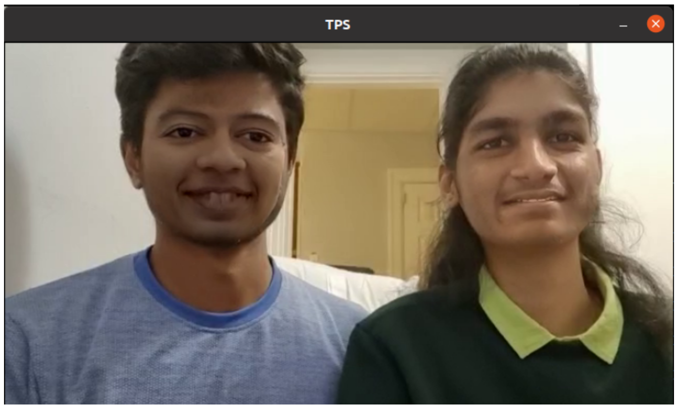

Fig. 30: Face swap using TPS of Anagha Mihir

H. Observations

The two major difficulties we observed were the number of triangles generated differently in triangulation and the speed of the video. We tried reducing the complexities and repetitive loops to reduce the overhead of the functions. A major change in the speed was observed. For the triangles issue, with indexing, the problem seemed to be solved. The faces are not warped efficiently as in some cases the prediction from dlib is disoriented. Thin Plate Spline seems to give better outputs, however there is not much difference in terms of visual appearance. When the 2 faces are of different complexion, the blend seems improper and it just appears to be one color on top of the other. A probable solution to this problem can be averaging the pixel intensities of the source and destination face areas (given by ConvexHull()) when blending the images. We have compared the outputs from all three different models using face similarity. we have also compared the results with some already deployed models available online.

II. Phase-II - MobileNet/3DDFA

In this Phase, we use an off-the-shelf model that uses the MobileNet architecture to give a full 3D mesh of the face in am image. The model also gives the 68 facial feature points similar to the dlib library. We have only used these 68 facial feature points and not the entire mesh and performed face swapping as in Phase 1. These 68 points are stored in a .txt file when we run their code as given. Thus, we use the text files generated for both the faces and extract the coordinates from them. These coordinates are then passed on to the TPS algorithm which works in the same way as before.

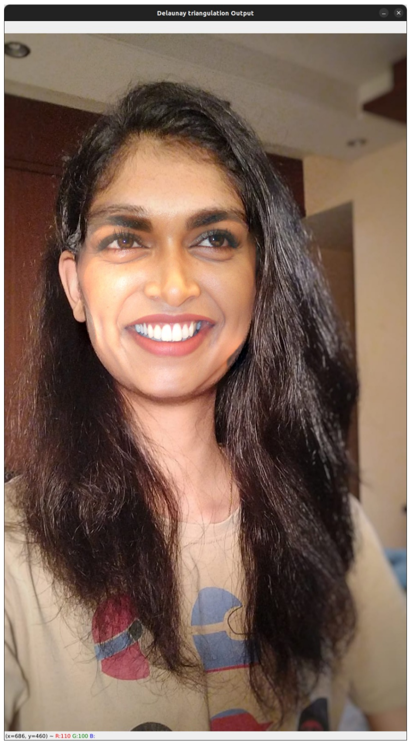
Delaunay triangulation Output
(x=686, y=460) ~ R:110 G:100 B:

Fig. 31: Phase 2 output of Anagha Deepika

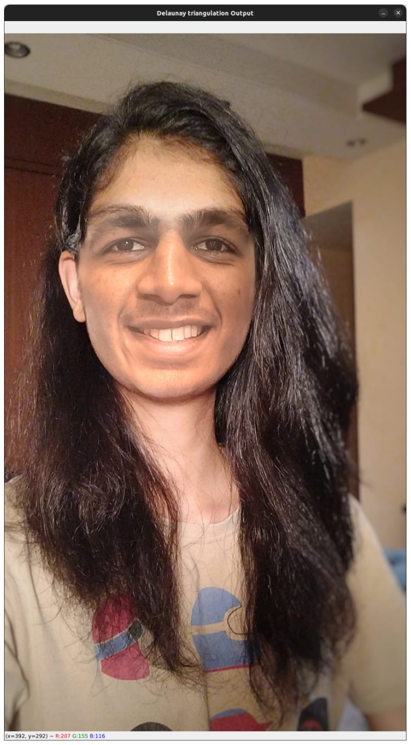
Delaunay triangulation Output
(x=392, y=292) ~ R:207 G:155 B:116

Fig. 32: Phase 2 output using TPS of Mihir Anagha

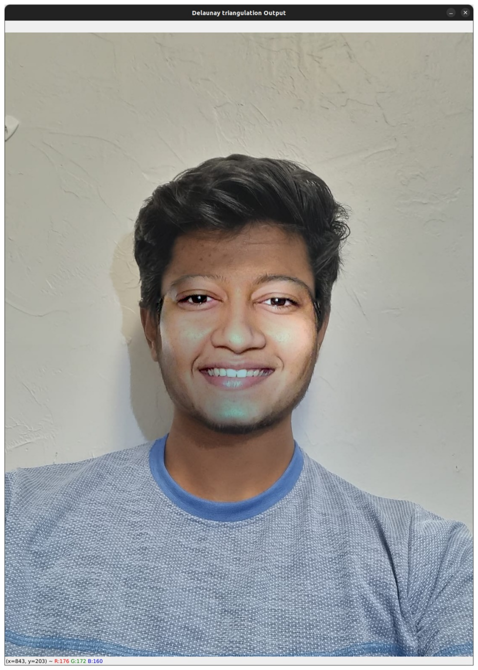
Delaunay triangulation Output
(x=843, y=203) ~ R:176 G:172 B:160

Fig. 33: Phase 2 output using TPS of Mihir Anagha

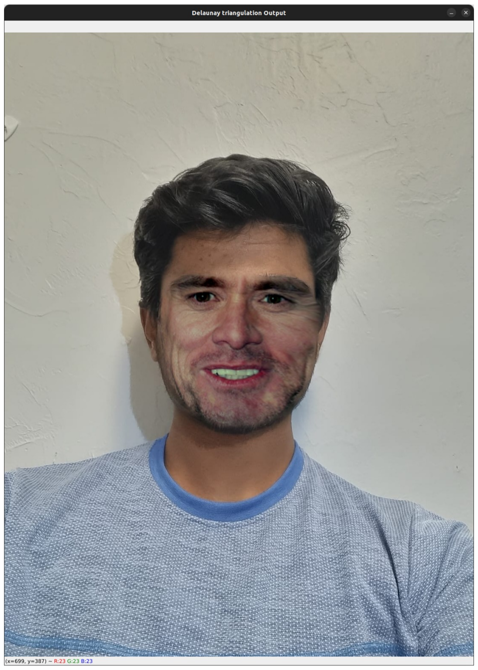
Delaunay triangulation Output
(x=699, y=387) ~ R:23 G:23 B:23

Fig. 34: Phase 2 output using TPS of Mihir Jim

III. Conclusion

Looking at all the results, the TPS output seems to be better than the delaunay. In some outputs, one triangle does not seem to be generated in the triangulation because of which the delaunay warping underperforms a little as compared to TPS. But, in certain images, like the Delaunay Output of Anagha & Deepika, the eyebrows appear to match properly which is not the case with TPS as well with Phase 2 outputs where the eyebrows are one below the other.

In Phase 2, as we are using the .txt file containing the facial feature coordinates, we were not able to use this algorithm for videos which requires generation of the 68 facial features for every frame. Thus, for us Phase 2 has been limited to just images, for which, it works quite well. We are also not using the 3D mesh given by the model to warp the image.

IV. References

https://khanhha.github.io/posts/Thin-Plate-Splines-Warping/
https://pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/