Project 2 - FaceSwap

Nikhil Mehra and Sayan Brahma
M.Eng Robotics
University of Maryland - College Park
nmehra@terpmail.umd.edu, sbrahma@terpmail.umd.edu

I. Introduction

The aim of this project is to implement an end-to-end pipeline to swap faces in a video just like Snapchat's face swap filter or this face swap website. It's a fairly complicated procedure and variants of the approach we'll implement has been used in many movies. Phase 1 of this projects deals with traditional face swap technique and Phase 2 deals with deep learning pipeline to swap faces. To draw a gist, given an image and a video stream, this project aims to implement an end to end pipeline to swap faces. We achieve this goal with traditional approach as well as with deep learning approach and then analyze and compare the outcomes from both the methods.

II. Phase 1: Traditional approach

The first step in the traditional approach is to find facial landmarks (important points on the face) so that we have one-to-one correspondence between the facial landmarks. One of the major reasons to use facial landmarks instead of using all the points on the face is to reduce computational complexity. Better results can be obtained using all the points (dense flow) or using a meshgrid. We implement image processing techniques using OpenCV and dlib. The overall pipeline is as follows: a. we read the image and detect the number of faces and then detect facial fiducials from the face. This is done for both, the source image and the target image. b. Triangulation followed by meshing is done based on the facial fiducial points obtained c. Warping each of these meshes from the source to their corresponding mesh in the target image. d. Once the warping is done, the cropped image is blended with the target image to get a smooth image i.e., the final outcome.

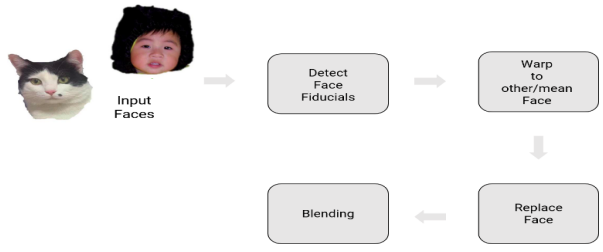


Fig. 1: Multiple Face Identified.

Following are the step wise explanation of the pipeline.

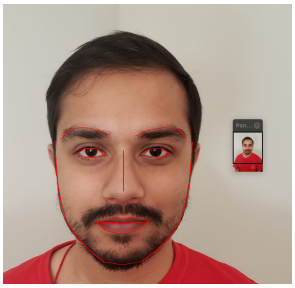
A. Input Image

The input images are first converted to Gray-scale. get_frontal_face_detector() and shape_predictor() functions from the dlib library are used which further detect faces in the input images. The function returns a list of 68 points each corresponding to a very specific landmark feature on the face. Due to the list of 68 features points from each image we get point correspondences for both the faces. Not all the points in the image are non-collinear to form triangles hence, we decided to only consider the points lying outside of the face. We do this by computing the Convex hull from all the points using the convexHull() function from OpenCV.

Fig. 2: Marked Facial features on multiple faces.

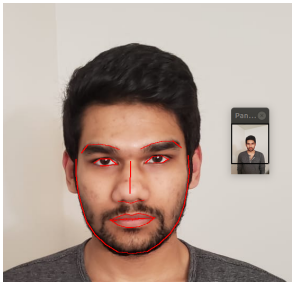
Fig. 3: Overview of face replacement lite pipeline.

(a) Good Features output



(b) Marked Feature Points

(a) Good Features output

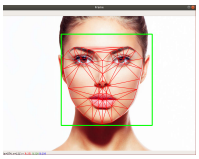


(b) Marked Feature Points

B. Face Warping using Triangulation

The facial landmarks obtained in the previous section act as the basis for the triangulation to be done in the next step. We used Subdiv2D package provided by OpenCV. The getTriangleList() funciton returns the coordinates of all the vertices of each of the triangles formed during triangulation. This process of triangulation is performed on both, the source and target images.

(a) Input image



(b) Triangulated image

(c) Voronoi image

After obtaining facial landmarks, we need to ideally warp the faces in 3D, however we don't have 3D information. Hence can we make some assumption about the 2D image to approximate 3D information of the face. One simple way is to triangulate using the facial landmarks as corners and then make the assumption that in each triangle the content is planar (forms a plane in 3D) and hence the warping between the the triangles in two images is affine. Triangulating or forming a triangular mesh over the 2D image is simple but we want to trinagulate such that it's fast and has an "efficient" triangulation. One such method is obtained by drawing the dual of the Voronoi diagram, i.e., connecting each two neighboring sites in the Voronoi diagram. This is called the Delaunay Triangulation and can be constructed in $O(nlogn)$ time. We want the triangulation to be consistent with the image boundary such that texture regions won't fade into the background while warping. Delaunay Triangulation tries the maximize the smallest angle in each triangle. We tried performing triangulation of the two images but in some cases the triangulation results were different i.e. triangle in one image didn't correspond to triangles in other image for corresponding features. To circumvent this issue we take the average of feature points and perform delaunay triangulation of them. Since we know the corresponding points which were averaged we get the same triangle correspondences in the same images. After triangulation, we have obtained the corresponding triangles in both the source and target images. We take advantage of the Barycentric coordinate system to find the transformation of each pixel from triangle in source image to the triangle in target image. The Barycentric points are represented by α, β, γ. Where A is source image and B is destination/target image.

$$\begin{bmatrix} \alpha_1 & \alpha_2 & . & . \\ \beta_1 & \beta_2 & . & . \\ \gamma_1 & \gamma_2 & . & . \end{bmatrix} = \begin{bmatrix} \mathcal{B}_{a,x} & \mathcal{B}_{b,x} & \mathcal{B}_{c,x} \\ \mathcal{B}_{a,y} & \mathcal{B}_{b,y} & \mathcal{B}_{c,y} \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_{\mathcal{B}_1} & x_{\mathcal{B}_2} & . & . \\ y_{\mathcal{B}_1} & y_{\mathcal{B}_2} & . & . \\ 1 & 1 & . & . \end{bmatrix}$$

We estimate the bounding box of each triangle and iteratively compute the Barycentric coordinates of each pixel and check if they lie inside the triangle or not. This is done by checking, if they lie inside the triangle or not. This is done by performing the following check:

$$\alpha \geq 0, \beta > 0, \gamma > 0, \alpha + \beta + \gamma \leq 1$$

If the pixel does lie within the triangle, with the help of barycentric coordinates, it's corresponding pixel location is found in the target triangle.

$$\begin{bmatrix} x_{\mathcal{A}} \\ y_{\mathcal{A}} \\ z_{\mathcal{A}} \end{bmatrix} = \mathcal{A}_{\Delta} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}$$

Here $\mathcal{A}_{\Delta}$ is given as follow:

$$\mathcal{A}_{\Delta} = \begin{bmatrix} \mathcal{A}_{a,x} & \mathcal{A}_{b,x} & \mathcal{A}_{c,x} \\ \mathcal{A}_{a,y} & \mathcal{A}_{b,y} & \mathcal{A}_{c,y} \\ 1 & 1 & 1 \end{bmatrix}$$

After we obtain $\begin{bmatrix} x_{\mathcal{A}} & y_{\mathcal{A}} & z_{\mathcal{A}} \end{bmatrix}^T$,these values are converted to homogeneous coordinates as follows:

$$x_{\mathcal{A}} = \frac{x_{\mathcal{A}}}{z_{\mathcal{A}}}$$

and

$$y_{\mathcal{A}} = \frac{y_{\mathcal{A}}}{z_{\mathcal{A}}}$$

To prevent holes in the image while warping, we use inverse transformation as opposed to forward transformation. However the results are still not satisfactory. Ideally, to get a nice image, we need to have as many triangles as possible. However, this is a workable approximation. Finally the target image is blended with the source image using seamlessClone function of OpenCV. The seamlessClone Since we are dealing with pixel level manipulation, this process is computationally intensive and takes a while to execute. Hence is not appropriate for this application.

C. Thin Plate Splines

In the previous section we used affine transformations to warp each corresponding triangle. This is not the best way to warp a human face due to its complexity. In this section we do the transformations using Thin-Plate Splines. Thin-Plate Splines are a spline based technique used for data interpolation and smoothing. The process is physically analogous to bending a thin sheet of metal. The goal is to find a mapping $f_x(x, y)$ and $f_y(x, y) : R^2 \rightarrow R^2$ such that:

$$f_x(x_i, y_i) = x_i^{'}$$

and

$$f_x(x, y) = a_1 + a_x x + a y_y y + \sum_{i=1}^{p} w_i U(||(x_i, y_i)(x, y)||_1)$$

The steps involve solving a system of linear equations of the form:

$$\begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_p \\ a_x \\ a_y \\ a_1 \end{bmatrix} = \left(\begin{bmatrix} K & P \\ P^T & 0 \end{bmatrix} + (p+3, p+3) \right)^{-1} \begin{bmatrix} v_1 \\ v_2 \\ . \\ . \\ . \\ v_p \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

where $K_{ij} = U(||(x_i, y_i)(x_j, y_j)||_1)$, and is of the size p × p. K is a symmetric matrix with the diagonal elements 0. K is plotted in the image shown below: $v_i = f(x_i, y_i)$ and i^{th} row of P is $(x_i, y_i, 1)$ and λ is a regularization term very close to 0. The second step is to transform the pixels of the face in the target image using the above developed TPS model and replace them in the source image. The weights obtained after solving the above set of equations are used to find the warped location of the desired point using the above equations.

D. Blending

For blending we used third party code, that corrects color of the warped face according to the destination image. We use cv2.seamlessClone for this operation. Figure 4 shows the effect after blending the target image with the destination image. The test sets given were passed through the given pipeline of TPS and the results are shown below.

E. Results

(a) Input Image 1

(b) Input Image 2

(a) Face Swaped using TRI

(b) Face Swaped using TRI

Fig. 4: Input Frame from the Video

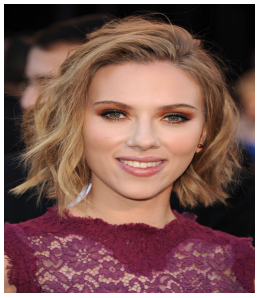


Fig. 5: Input Image

Fig. 6: Test 3 Frame 92 TRI Output

Fig. 7: Test 3 Frame 92 TPS Output

Fig. 8: Input Frame frome the Video

Fig. 9: Input Image

Fig. 10: Test 1 Frame 88 TRI Output.

Fig. 11: Test 1 Frame 88 TPS Output.

Fig. 12: Input Frame from the video

Fig. 13: Test 2 Frame 28 TRI Output.

Fig. 14: Test 2 Frame 28 TPS Output.

F. Discussion/Conclusion for Phase1

Each test set was aimed towards testing different aspects of the pipeline. The test 1 output is good in terms of raw facial landmark detection in each frame. There is less motion blur as compared to other test videos. Test 2 was to swap the two bigger faces in the image. The pipeline also gives satisfying results for this test set. The only problem we face is that facial landmark is not very robust. In some frames when only one side of the face is visible the dlib facial detection fails. A similar case occurs for very illumination of the face in test 3 video. The outputs for these videos can be improved by implementing a motion filtering or feature tracking algorithms like EKF(Estimated Kalman Filter) on top of the current pipeline.

III. Phase 2 - Deep learning Approach

For this phase, an off-the-shelf model is trained to obtain face fiducials using deep learning. The approach used here implements a supervised encoder-decoder model to obtain the full 3D mesh of the face. We ran the code to obtain face fiducials/full 3D mesh.

A. Results

(a) Face swaped using PRNet

(b) Face swaped using PRNet

Fig. 15: Test 3 Frame 92 PRNet Output

Fig. 16: Test 1 Frame 88 PRNet Output.

B. Discussion/Conclusion for Phase 2

Deep Learning approach, PRNet provides way better results, with less flickering and good color blending. The results of Triangulation are, however, not as satisfactory and the face warping is not up to par as compared to the deep learning approach. The process of Triangulation is piece-wise linear and assumes the transformation to be planar, whereas the neural net outputs a smooth face warp created from a 3D reconstructed face based on it's learned features. Between TPS and PRNet, PRNet gives better results accounting for different face alignments and illuminations. However, the computation time required for PRNet is much more that what required for TPS. Also, close results can be obtained by applying edge dilution and seamless color morphing on the top of of TPS output.

REFERENCES

[1] GitHub - Face Swap *https://github.com/wuhuikai/FaceSwap.*

[2] Joint 3D Face Reconstruction and Dense Alignment with Position Map Regression Network *By Yao Feng, Fan Wu, Xiaohu Shao, Yanfeng Wang, Xi Zhou (Submitted on 21 Mar 2018)*

[3] Gianluca Donato and Serge Belongie. Approximate thin plate spline mappings. *In Proceedings of the 7th European Conference on Computer Vision-Part III, ECCV '02, pages 21–31, Berlin, Heidelberg, 2002. Springer-Verlag*