Face Swap

Shounak Naik
Robotics Engineering Department,
Worcester Polytechnic Institute,
Worcester, MA, USA.
ssnaik@wpi.edu

Venkatesh Mullur
Robotics Engineering Department,
Worcester Polytechnic Institute,
Worcester, MA, USA.
vmullur@wpi.edu

I. Introduction

This reports presents a detailed walk through 2 methods of doing FaceSwap. FaceSwap is used heavily in the Augmented Reality world with the most popular use by Snapchat. The 2 methods presented in this report are FaceSwap by using 1) Delenauy Triangulation and 2) Thin Plate Spline. We also compare the results of these 2 methods.

II. Delanauy Triangulation

A. Landmark Detection

We know that all human faces common feature points such as eyes, lips etc. We use this fact as a starting point for both the methods of FaceSwap. These feature points are used to establish 1:1 correspondences between faces. We have used the bulit-in functions inside the `dlib` library to achieve these feature landmark points inside the face. The `dlib` library used HoG detectors to detect the face. After we detect the face use detect the face landmark points. An example of this face detection can be seen in Fig 1

Fig. 1. Landmark Detection using dlib

B. Triangulation

After getting the landmark points in the face. We use these points to form the Delenauy triangulation. This triangulation is formed such that the smallest angle is maximised in each triangle. We assume that these triangles represent a plane between them. Thus the 3D face structure is approximated as a set of 2D planes. We use the fact that the triangles formed between 3 landmark points correspond to each other across faces. We have used the *cv2 getTriangleList* function to get the triangulation. Using this function on two faces does not guarantee same number of triangles and thus a a 1:1 correspondence between the triangles in the 2 faces. To overcome this, we have marked each triangles corner coordinates on one face and then constructed triangles on the second face using this information. Each landmark point is indexed and thus it is easy to map corresponding triangles using the earlier technique. This ensures a 1:1 triangle correspondence across faces. The triangulation is visualized in Fig 2. Correspondence between triangles is achieved by the information seen in Fig 3.

C. Face Warping using Triangulation

We use inverse warping to warp the faces into one another. We use this so that we don't have holes in the warped image. We do warping by doing the following for each triangle.

1) For each triangle in the target face B, the Barycentric coordinate for every pixel are calculated using the following equation.

$$\begin{bmatrix} \alpha_1 & \alpha_2 & . & . \\ \beta_1 & \beta_2 & . & . \\ \gamma_1 & \gamma_2 & . & . \end{bmatrix} = \begin{bmatrix} \mathrm{B}_{a,x} & \mathrm{B}_{b,x} & \mathrm{B}_{c,x} \\ \mathrm{B}_{a,y} & \mathrm{B}_{a,y} & \mathrm{B}_{c,y} \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \mathrm{x}_1 & \mathrm{x}_2 & \mathrm{x}_3 \\ \mathrm{y}_1 & \mathrm{y}_2 & \mathrm{y}_3 \\ 1 & 1 & 1 \end{bmatrix}$$

Where α, β ,and γ are Barycentric points. Here we use the fact that each point between 2 corresponding triangles have the same barycentric coordinates. Here a, b and c are triangle coordinates.

2) We can now determine the source points using the calculated barycentric points. We know that a point (x,y)

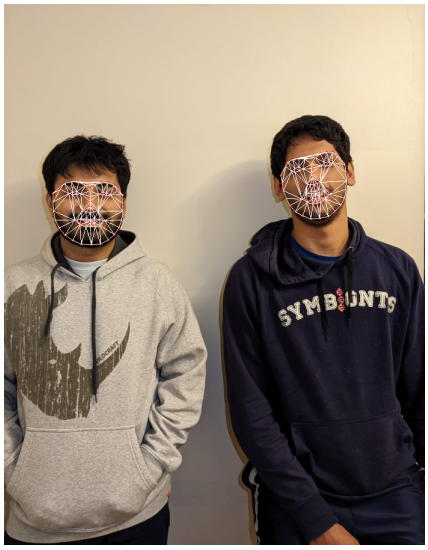

Fig. 2. Delanauy Triangulation

Fig. 3. Face Landmark indexes

is inside the triangle if :

a) $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$, $0 \leq \gamma \leq 1$
b) $0 \leq \alpha + \beta + \gamma \leq 1$

3) We can now compute the corresponding source pixels in Face A. It is done by the following equation:

$$\begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix} = \begin{bmatrix} \mathrm{A}_{a,x} & \mathrm{A}_{b,x} & \mathrm{A}_{c,x} \\ \mathrm{A}_{a,y} & \mathrm{A}_{a,y} & \mathrm{A}_{c,y} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \beta_1 \\ \gamma_1 \end{bmatrix} \quad (1)$$

where the A matrix is the source face matrix.

4) We can now compute x_A and y_A by using the following homogeneous rule: $x_A = x_A/z_A$ and $y_A = y_A/z_A$.
5) Now we go to the x_A,y_A pixel and then copy the pixel value and write this value into the destination pixel x_B,x_B. Since the x_A,y_A can be in between points, we use the `interp2d` method to interpolate the pixel value.

An implementation trick that we used is to add a λ Identity matrix to the B matrix above before finding its inverse. This ensures all inverses don't have large values. Without using this λ , we get a pixelated image. Debugging this led us to realize that using λ is a must here. We have used the `cv2's seamlessClone` function to blend the faces together. This adjusts the illumination differences among faces. The warped image with the FaceSwap is seen in Fig. 4

Fig. 4. Face Swap by Triangulation

III. Thin Plate Spline

In the previous section, we used triangulation method to get the 3D information using an image. Each triangle represents a 2D plane and this is done on both the faces. Basically we are using affine transformation on the faces. Since human faces are complex and have a smooth surfaces, a more elegant way to warp the faces would be using thin plate splines. Thin plate spline is basically a spline-based interpolation technique that uses radial basis function to warp the images.

1) Radial Basis Functions: Radial Basis Functions are real valued function whose value is only dependant on the distance from any arbitrary point. Given a set of control points, the function maps any location to some other shifted location. The farther the point from the control points, the lesser is the effect of that point to the value. Ideally, Gaussian and $r^2log(r)$ are basic functions used as radial basis functions. Using radial basis functions, we are going to warp the face mask in such a way that the pixels away from the landmark points are less affected by the shifting of pixels and the nearer pixels are more affected.

Basically, considering 2 faces in the image, the control points of first images are shifted in correspondence with the second and vice-versa. Equation 2 shows the Radial basis function this paper uses.

$$U(r) = r^2 log(r^2) \quad (2)$$

$$where, r = \sqrt{x^2 + y^2}$$

$$\begin{bmatrix} K_{00} & K_{10} & K_{20} & 1 & x_0 & y_0 \\ K_{01} & K_{11} & K_{21} & 1 & x_1 & y_1 \\ K_{02} & K_{12} & K_{22} & 1 & x_2 & y_2 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ x_0 & x_1 & x_2 & 0 & 0 & 0 \\ y_0 & y_1 & y_2 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ a_0 \\ a_x \\ a_y \end{bmatrix} = \begin{bmatrix} x'_0 \\ x'_1 \\ x'_2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Fig. 5. Forward System of Equations

2) TPS Inverse Warping: Our primary aim is to compute a TPS that maps from the feature points in face2 to the corresponding feature points in face1 and vice-versa. The main TPS function is defined by the equation 3.

$$f_x(x_d, y_d) = a_1 + (a_x)x + (a_y)y + \sum_{i=1}^{p} \omega_i U(\|(x_i, y_i) - (x, y)\|) \tag{3}$$

When we write this expression as a system if equations, we get the following:

$$\begin{bmatrix} K & P \\ P^T & O \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ . \\ . \\ \omega_p \\ a_x \\ a_y \\ a_1 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ . \\ . \\ v_p \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

where K is the matrix that basically used the radial basis function. This is a symmetric matrix with the distances from the control points.

$$K = \begin{bmatrix} 0 & U(R_{12}) & U(R_{13}) & ... & U(R_{1p}) \\ U(R_{21}) & 0 & U(R_{23}) & ... & U(R_{2p}) \\ ... & ... & ... & ... & ... \\ U(R_{p1}) & U(R_{p2}) & U(R_{p3}) & ... & 0 \end{bmatrix}$$

P is the matrix having source coordinates. Basically i^{th} row of the P matrix is defined as (1,x_i,y_i).
Basically we want to warp the control points of the destination corresponding to the source. For that we can try this forward warping, but since forward warping has many issues like it can it can have holes and blobs in between. To counter that, we usually do inverse warping so that we can interpolate the pixel values at destination and then get these values to put them in the source face to swap them. The system of equation after substituting the K and P values look like the figure 5

To solve this, we must generate a inverse system of equations. It is explained in the equation 5

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ . \\ . \\ \omega_p \\ a_x \\ a_y \\ a_1 \end{bmatrix} = \left(\begin{bmatrix} K & P \\ P^T & O \end{bmatrix} + \lambda I(p+3, p+3) \right) \begin{bmatrix} v_1 \\ v_2 \\ . \\ . \\ v_p \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

Fig. 6. With λ = 1

Fig. 7. Face swap With λ = 50

Here, the lambda term is supposed to be a parameter so that the determinant does not go to zero. Hence, inverse of this matrix is valid. Previously, we assigned the value of λ as 10^-6. We did not get a satisfactory result as the mask was all white. This problem is shown below in the figure 6
Equation 5 has to be done for both x and y coordinates.

- In K, P we put the coordinates of source.
- v_i = either $xorY$ coordinates of destination.
- We get the weights that transitions from source to destination.
- Now, using these weights, put them in equation 3.
- In 3, put x, y as all points of the source, x_i, y_i as the control points in the source to get F_xorF_y depending on the v_i's
- Now these are the point that map from destination to source, so put these coordinates in the source face to get a face swap.

The output we got is shown in the figure 7

IV. Conclusion

Theoretically, the TPS method should work better than the triangulation as the triangulation has some major assumptions about the planarity of the face. But in our implementation we found that the triangulation method works more reliably. Also the TPS is computationally slow since we had to calculate the TPS values for each pixel. In triangulation, we used a matrix multiplication operation to get the desired result hence decreasing runtime. Our implementation works for any image containing 2 faces. This does not include swapping faces across 2 seperate faces. Although this can be acheived by making trivial changes to the code.

References

[1] https://khanhha.github.io/posts/Thin-Plate-Splines-Warping/

[2] https://cmsc733.github.io/assets/2019/p2/results/pdf/Abhi1625$_p$2.*pdf*