Automatic Seamless Face Replacement in Videos

Yiren Lu, Dongni Wang
University of Pennsylvania
December 2016

1 Introduction

In this project, the goal is automatic face detection and replacement in videos. Given a video, we first detect faces and extract critical facial feature and contour coordinates from every frame of the given video, using a 3rd party library, *dlib*. Then, we run through a process pipeline to achieve a seamless face replacement result on each of the frames. Finally, we generate a video with processed frames, in which all chosen faces are replaced by our pre-generated source faces of choice.

Our team choose **Option 2**, and we aim to achieve satisfactory overall results for seamless face replacement with the help of standalone 3rd party libraries.

2 Pipeline Overview

Our pipeline takes two video inputs, namely the given target video in which target faces would be replaced and a source video clip which provide the source faces. The output of our pipeline is one video with the same number of frames as the given target video, with one or all faces replaced by a similar face detected in the source video. Normally, source faces are generated in images. However, to choose the best source faces that are most similar to our target faces in terms of shape and pose, and furthermore, to avoid the jiggling of faces resulted from per-frame wrapping, our team choose to generate source faces directly from videos.

The first step in the pipeline is to take the videos and pass them to textitdlib for the face detection, the results include the critical facial feature and contour coordinates for all the faces from every frame. Then, if faces are detected, we would start/update the face tracking using KLT, otherwise we would use the face tracking results to generate estimated facial feature and contour coordinates. After this step, we smooth the face-region images and conduct shape and episode matching, to find the most similar 20-frame slices in source and target videos. More details of the matching method will be discussed in the next section.

After we find the source faces, we would perform the face wrapping using TPS and face replacement using masks generated by either convex hull or seamless carving in log-polar space. Next, we do the Poisson blending to adjust the color of wrapped faces and finally, generate the resulting video.

3 Method

3.1 Face Detection

For the task of face detection. Our team first experimented with *Matlab* built-in *CascadeObjectDetector* and tune the parameters to get benchmarks on video frames. However, the precision of this detector is not promising. Tuning the thresholds does not seem to improve overall detection performace. We turned our eyes to third party libraries for more reliable detection results then.

The first third party detection method we tried is *Facepp*. Compared with the built-in detector, this method significantly improves prevision and achieve better over detection accuracy. What's more, the facial landmarks were also detected, which could be of help to get the face region for carving. One con for this method that it is slow because we are not provided with the source code, but rather submitting detection jobs online. Each 500x500 frame takes at least a few seconds. We first relied on this detection method to proceed with our pipeline implementation, until later we found a open source library which provides us with comparable detection accuracy while being significantly faster.

Our final face detection employed the *dlib* library. As

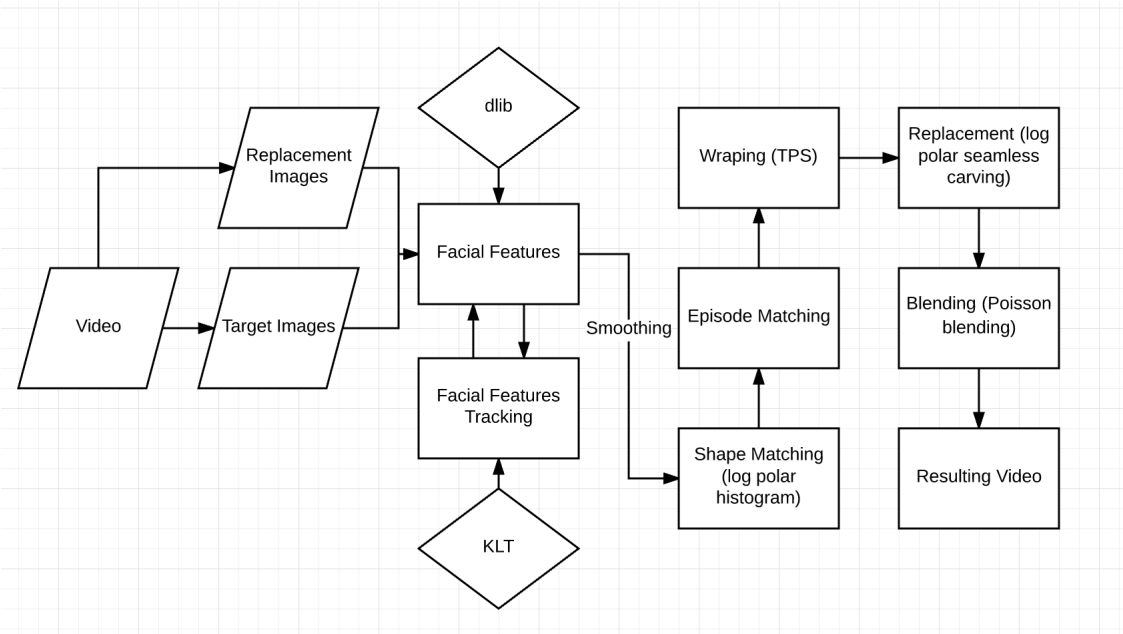


Figure 1: Pipeline

stated earlier, this method performs well both in terms of speed and accuracy. *dlib* a library most written in C++ and comes with Python API. We wrote a python wrapper for this task to further process the results and export them in the desired *.mat* format.

Figure 2: Face detection results by Face++

3.2 Face Tracking & Trajectories Smoothing

Results from dlib face detection are pretty good. However, for some challenging cases (lighting changes/boundary head angles), dlib failed to detect faces. We filled in the detection failure frames by a KLT tracker. If the face detector failed for a consecutive number of frames (e.g. 10 frames), the tracking results would go wild. In this case, we stop the tracking and wait for the next successful detection.

The problem in directly using frame-by-frame detection to perform replacement is that the detection results between frames are completely independent, therefore, they tended to jitter a lot, which results in unstable face warping and blending. To achieve stable tracking trajectories, after tracking, we applied a 1-d "temporal Gaussian convolution" to each landmark point's trajectory. This smoothing scheme resulted in nature and realistic face landmark trajectories.

$$t(i) = \sum_{k} t(i-k) * g(k)$$

Where, t is the trajectory of one landmark, g is a 1-d Gaussian kernel, and the range of k controls the span of the convolution.

3.3 Face Matching

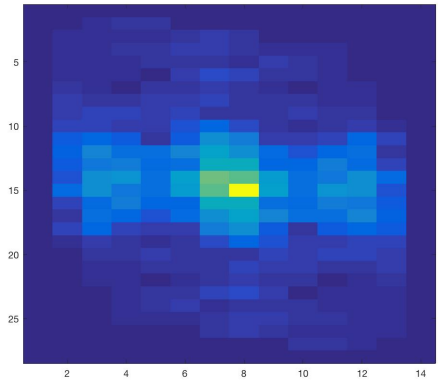
In video, for most times, the pose of the target face (the face to be replaced) does not completely agree to the source face (the face for replacement). This results in unnatural face replacement results. To address this problem, we implemented landmarks shape matching to match similar pose of faces for replacement.

3.3.1 Face Shape Features

Feature extraction of the target face follows [2].

- Convert the landmark points into log-polar coordinates
- For each landmark point, compute the relative coordinates of the remaining points.
- Generate a histogram of all the relative coordinates.

A visualization of one face feature is the following:



There are a few good properties of this feature:

- Translation invariant: since it computes the relative coordinates
- Rotation invariant: subtract an offset angle of vector between two eyes' center.
- Scale invariant: we normalize the radius feature by dividing the largest radius (The paper used median of radius, but we found it to be 0).

To match the features, we can directly use euclidean distance or using χ^2 distance:

$$\frac{1}{2}\sum_{k=1}^{K}\frac{[g(k)-h(k)]^2}{g(k)+h(k)}$$

In practice, we found that euclidean distance actually worked better for our task.

3.3.2 RANSAC Episodic Matching

The matching algorithm looks for the best matching for each frame. However, due to head angle changes, the lighting conditions of matched source faces change a lot. This results in flashing in the final replacement result. To address this problem, we adopted episodic matching. The basic idea is to match a sequence of consecutive frames (e.g. 20 frames) instead of a single frame. To implement this, we can employ RANSAC framework. For each eposide:

1. Loop N iterations
2. Randomly pick a consecutive sequences of same length in the source video
3. Compute error for each pair of corresponding frames.
4. Count inliers with a threshold e
5. Keep the matching with largest number of inliers

Here, we just used euclidean distance, but other metrics are also worth trying.

3.4 Face Replacement

The matching algorithm provides us with the source faces that are most similar with their associated target faces. Now, we could proceed to the next step in pipeline: face replacement.

3.4.1 TPS Face Warping

Our team decide to use thin plate spline (TPS) warping for this task. Compared to the affine transformation, TPS usually gives smoother curves around the carving edges, the resulting warping image does confirm our statement. Specifically, since TPS is applied to coordinate transformation, we use two splines, one for the displacement in the x direction and one for the displacement in the y direction. We then transform all the pixels in target faces by the TPS model, and read back the pixel value in source faces directly.

3.4.2 Log-Polar Carving

To automatically select the face region for replacement, we implement the seamless carving algorithm in log-polar coordinates. The reason that we transform the image from using Euler coordinates to log-polar coordinates is that we want to find a seam carving around the face, which should also form a closed cycle in Euler coordinates to make a valid mask.

The algorithm details as follows:

1. Set the origin of the log-polar coordinates to be the detected position of the nose (center of the face). Set the radius of the region to be considered to be the distance between the center of the face and the furthest contour points from the center of the face. Set the number of wedges to be 360 and the number of rings to be considered equal to the radius*2.
2. Reorganize the approximate face region in log-polar coordinate using the parameters set earlier.

3. Find the minimum-energy (we also tried edge map, besides energy map) seam carving of the reorganized image. To force it to return a cycle in the Euler space, we must ensure the first and the last angle should have the same carving radius. In other words, the starting and ending column (ring) has to be the same for finding a vertical seam from the first to the last row (wedge). We can achieve this by making all the other options very expensive (cost high energy).

4. We could also force the seam to go through other contour points returned by the detector. However, we need to be careful since the dynamic programming algorithm we implemented in calculating the min seam would only consider three possible positions in the next level. If the contour points to be crossed are too close together, it can potentially affect the returned seam negatively.

5. Lastly, after getting the seam carving in the log-polar space, we transfer the coordinates back to the image grid. We will then have a complete cycle which is the face region to be replaced.[1]

Figure 3: Min-energy seam carving returned by the log-polar carving algorithm

3.4.3 Poisson Blending

After we have the wrapped face and carving face region ready, we want to seamlessly blend the wrapped face into the target image. Poisson Blending is a natural fit for this task. The key idea of the Poisson blending (gradient domain blending) is to apply the gradient of the source image (wrapped face) to the target image's replacement pixels (carved region), but keep boundary pixels unchanged. [3]

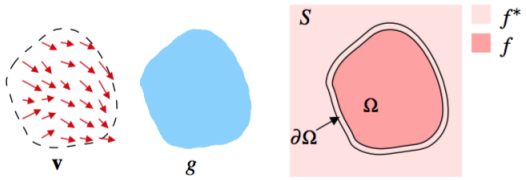


Figure 1: **Guided interpolation notations**. Unknown function f interpolates in domain Ω the destination function f^*, under guidance of vector field $\mathbf{v}$, which might be or not the gradient field of a source function g.

4 Implementation

All our code was fully vectorized. To save runtime, we re-sized the test video from 1280x720 to 640x360. On this resolution, our algorithm achieved 0.8-1.5s per face at runtime depending on the size of the face. (Detection was done offline by dlib).

5 Results & Discussion

Below are some face replacement results. Our algorithm yields pretty desirable results both in static image and in video. (video results were attached in the submission). Here are some main factors that we think contributed to the final results:

- Good face landmarks detection. We have to admit that our pipeline built up on the good face landmarks detection provided by dlib.
- Trajectories smoothing made the face replacement results very stable.
- Face episodic matching selects the most similar sequence to the target face episode. This compensates the head pose changes.
- Seamless poisson blending helps blend the source face with the target face.

There are two aspects that we can improve base on the current pipeline:

- Refined face carving with GMM. We can actually use GMM to do face pixels classification. To do so, we can crop the detected face range and train a GMM online, and then use GMM to generate a face likelihood map. With proper threshold, we can obtain another face mask. Along with the previous convex hull and the log-polar carving, we can generate a refined face mask. This can be effect since

including (even a small amount of) background pixels into the carving is damaging to the final blending. GMM can rule out the background pixels.

- Head pose estimation by PnP. There are mature technologies to estimate the head pose base on 2d images. These methods can improve episodic matching.

Figure 4: Left:original face; right: replaced face

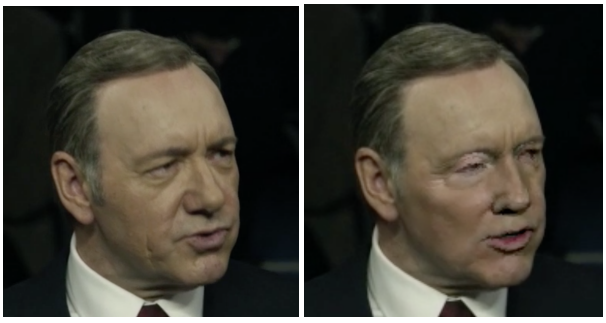


Figure 5: Side angle. Left:original face; right: replaced face

Figure 6: Side angle. Left:original face; right: replaced face

References

[1] AVIDAN, S., AND SHAMIR, A. Seam carving for content-aware image resizing. In *ACM Transactions on graphics (TOG)* (2007), vol. 26, ACM, p. 10.

[2] BELONGIE, S., MORI, G., AND MALIK, J. Matching with shape contexts. 81–105.

[3] PÉREZ, P., GANGNET, M., AND BLAKE, A. Poisson image editing. In *ACM Transactions on Graphics (TOG)* (2003), vol. 22, ACM, pp. 313–318.