CIS 581 Project 4C: Face Replacement

Professor Jianbo Shi
Adnan Jafferjee | En Hui Zou | Daniel Harris

Introduction

For this project we chose to attempt track 2 whereby we were able to use any open source libraries found online. While several libraries were found, the Dlib and CV2 libraries contained the largest suite of functions, which yielded the best results for feature detection and gradient domain blending. The primary goal of this project was to seamlessly swap a face from another video or image onto another face in a target video. Success was measured based on how visually realistic the resulting face swap videos were. Critical elements to achieving this were synchronising movement and facial expressions of the tracked faces throughout the video as well as seamlessly blending the transitions between the swapped faces.

Methodology

1. *Contrast-Limited Adaptive Histogram Equalization*

Normal histogram equalization is a correction applied for videos with changing illumination. This method generates a histogram representing the distribution of pixel intensities in the image. After doing so, the most frequent intensity values are spread out so that regions of lower local contrast can gain a higher contrast. This however, only works well for images where the pixel intensity distribution throughout the image is relatively small. Contrast-limited adaptive histogram equalization is a very similar process but instead of generating a single histogram representing for the entire image, it generates several histograms for the pixels intensity distributions around several pixels and distributes the most frequent pixel intensities amongst each of these local regions. This is illustrated in *Figure 1* below:

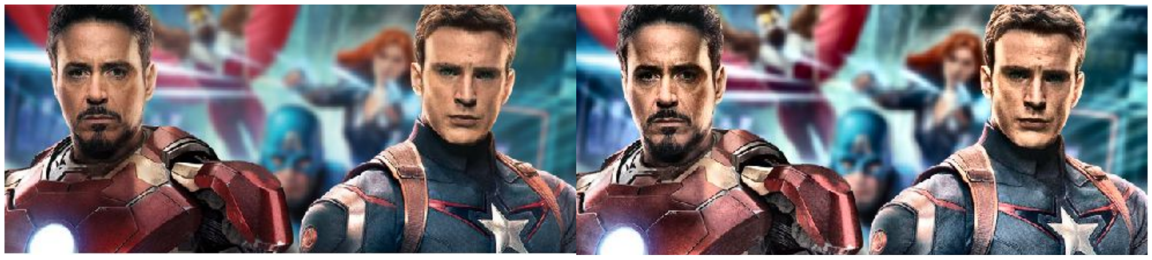


Figure 1: Effect of CLAHE

After resizing each frame to ensure they were of the same resolution, this process was applied at the beginning of each frame in an attempt to eliminate large discrepancies in contrasts throughout the image. This was specifically helpful when later using gradient domain blending and resulted in much better results for the seamlessness of the face swap.

2. *Detecting Facial Landmarks*

The next step in successfully swapping faces was determining a sufficient number of feature points to determine the various facial landmarks such as: the face, nose, eyes, ears and lips. Dlib's library for frontal face detection was incredibly helpful for this portion of the project as it almost always reliably found 68 feature points. These points were found on both the source image and the target image and used to generate a convex hull mask.

3. *Convex Hull Face Alignment*

The convex hull mask was important when trying to align the two faces. Luckily, OpenCV's library contains a convex hull function, "cv2.convexHull," that takes in the detected feature points as an input and returns the indices of all the pixels inside the outermost bounds of the feature points for both images. This was especially useful when ensuring that both masks were of the same size to avoid any issues when trying to overlay the source image face onto the target image.

4. *Warping using Delaunay Triangles*

Using the same feature points from Dlib's frontal face recognition function, two more functions, "calculateDelaunayTriangles" and "warpTriangle," functions were used to warp the face of the source image to ensure that the facial landmarks would line up with the facial features of the target image.

5. *Seamless Cloning using OpenCV*

Once the source image face was successfully warped and overlaid onto the face of the target image, we used gradient domain blending (a.k.a poisson blending) from the cv2 library to smooth out the contrast between the overlaid source image and target image. This allowed us to achieve more realistic transitions between the skin tones of the faces being swapped.

6. Optical Flow for Feature Tracking

One artifact that we noticed in our videos was the occasional re-emergence of the original face in the middle of the video. We hypothesize that this occurs in frames where our face detector is unsuccesful in detecting a face, thus a convex hull is not generated to accommodate a source face. One solution to this problem, which we implemented was to calculate the optical flow between frames whenever a face isn't detected. This allows us to map the transition of the face between frames based on a predicted displacement of key features (in this case, the corresponding facial landmarks between frames).

The complete process of face swapping is illustrated visually below in *Figure 2:*

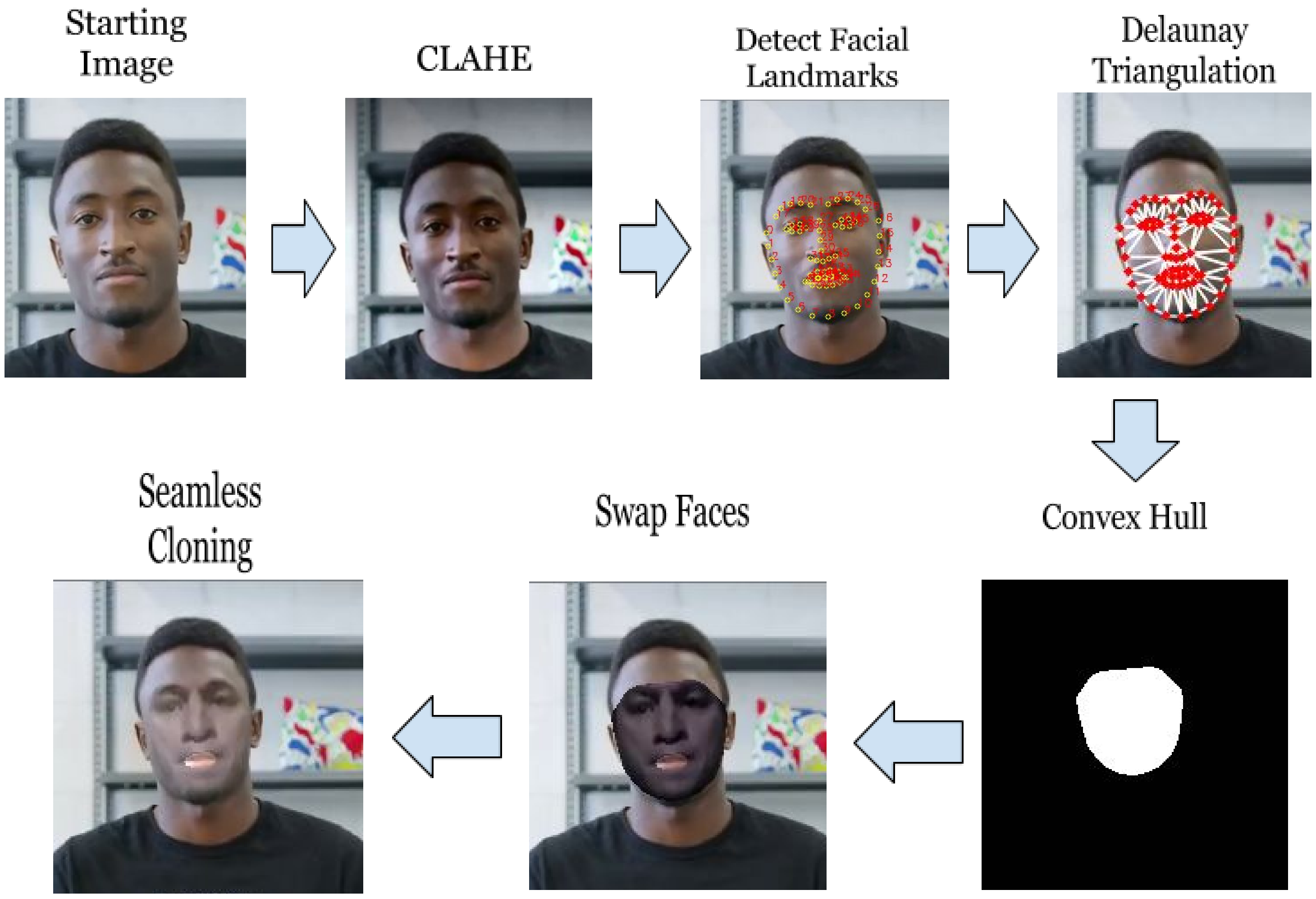
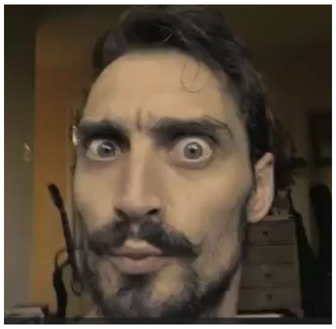
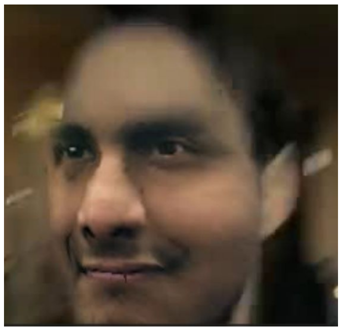
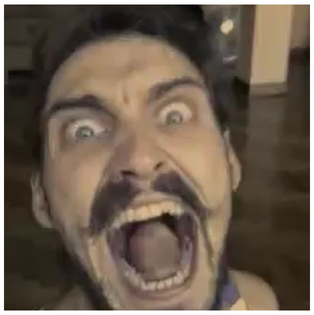


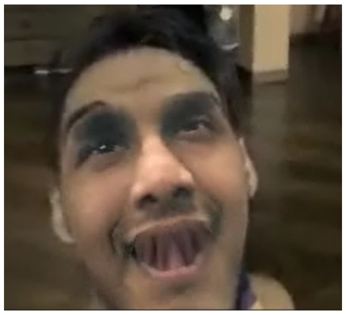
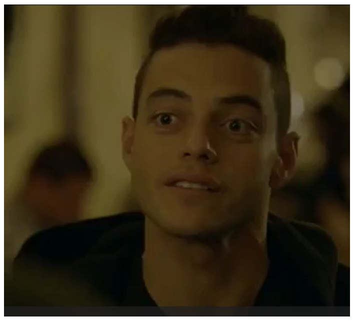
Figure 2: Pipeline for Face Swapping

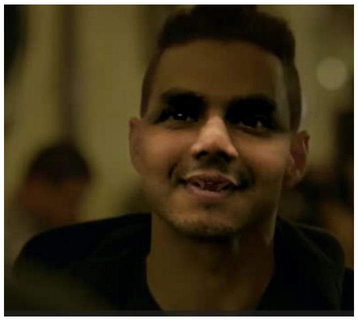
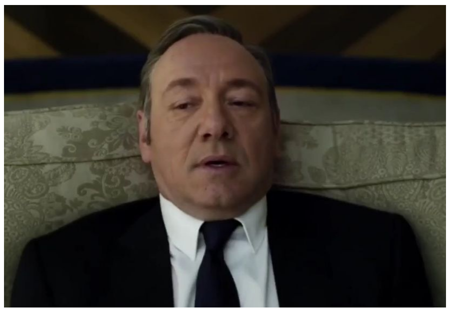
CIS 581 Project 4C: Face Replacement
Method: Track 2

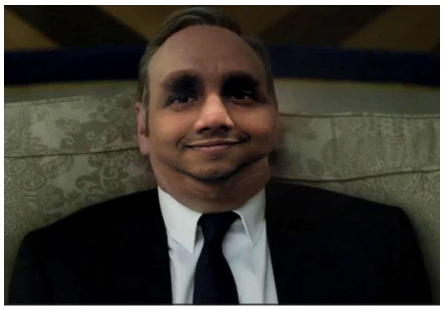
Results



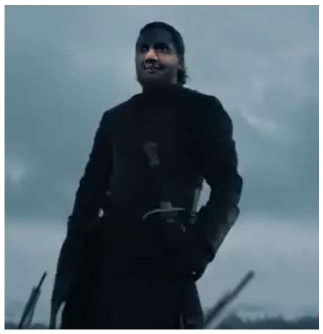



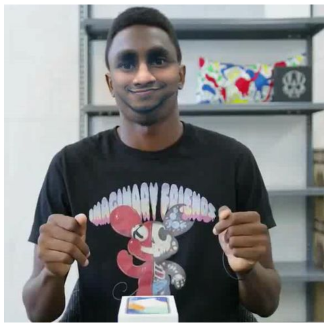



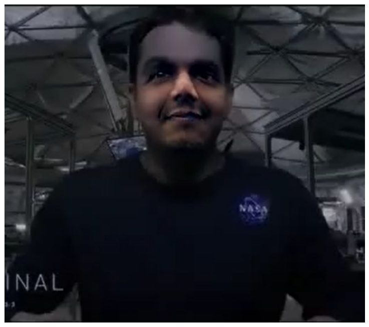



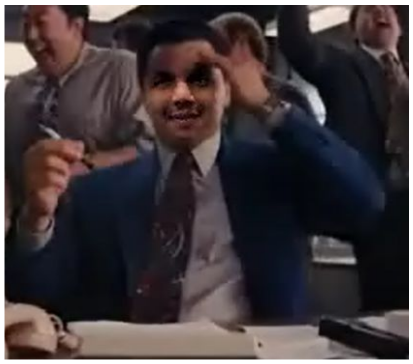


CIS 581 Project 4C: Face Replacement
Method: Track 2









Evaluation and Future Work

Cost Function for Mouth Feature Matching

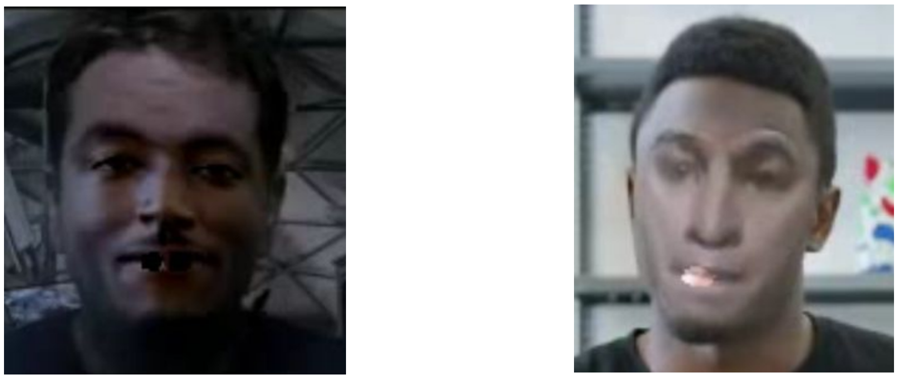


Figure 3: Artifacts in the Mouth Area

Another artifact we encountered was the presence of missing pixels around the mouth area of the face swapping targets. This can be seen above in *Figure 3*. One potential reason for this is that the videos are not well-synchronized with respect to the movement of the subjects mouths. As the relative movement of the mouths in these videos is significantly more than other parts of the faces, this area is especially susceptible to poor feature matching.

One paper that we found on the subject (by Sunkavalli et al.) addressed this issue by minimizing a cost function of the Euclidean distance between the vertices of the two subjects upper and lower lips. Using this cost function, they were able to map frames from a given pair of videos that best matched the subject's mouth motions. One major drawback of this method is that it fails to maintain temporal consistency, which makes real-time swapping (an ultimate goal) an impossibility.

Head Orientation Detection and Correction

Our code is not robust in dealing with situations where the orientation of the face deviates dramatically from the front-facing position. This is because we are using Dlib's face detection, which was trained on datasets primarily consisting of front views of faces. One potential solution to this problem is referenced by Kazemi and Sullivan, who suggest using the Jacobian to compute the orientation vector of the face (as demonstrated by *Figure 4*). By doing this, we can predict when the facial landmark detector doesn't detect a face, and account for that using optical flow.

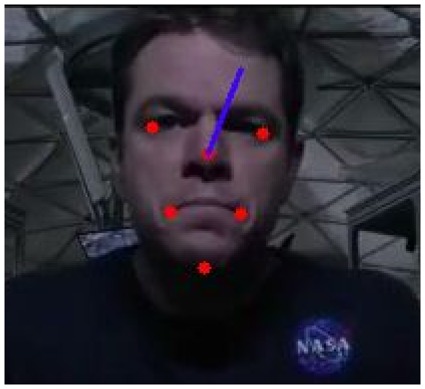


Figure 4: Detecting Orientation of the Face using Jacobians

References

[1] dlib C Library. [Online]. Available: http://dlib.net/face_detector.py.html. [Accessed: 17-Dec-2017].

[2] S. Mallick, "Home," Learn OpenCV, 05-Apr-2016. [Online]. Available: https://www.learnopencv.com/face-swap-using-opencv-c-python/.
[Accessed: 17-Dec-2017].

[3] Spmallick. "Spmallick/Learnopencv." GitHub, 9 Dec. 2017, github.com/spmallick/learnopencv.

[4] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014.

[5] K. Dale, K. Sunkavalli, M. K. Johnson, D. Vlasic, W. Matusik, and H. Pfister, "Video face replacement," Proceedings of the 2011 SIGGRAPH Asia Conference on - SA 11, 2011.