Dynamic Lip-Flip Application

Trisha Lian
School of Electrical Engineering
Stanford University
Email: tlain@stanford.edu

Kyle Chiang
School of Electrical Engineering
Stanford University
Email: kchiang@stanford.edu

***Abstract*—With the growing popularity of video calling, there comes an opportunity to develop fun special effects to use on chatting platforms. Our goal for this project was to create "Lip-Flip", a real-time app that swaps the mouths of two users sitting in front of two webcams. Realism is achieved by having inserted mouths dynamically match the movement and scale of each face. First, we apply color and brightness calibration in order to reduce any differences between the two webcams. Next, we detect the mouth locations using the Viola-Jones algorithm and OpenCV-supplied Haar cascades. Lastly, we use Laplacian pyramid blending to realistically insert new lips onto a user's face. Because the app runs in real-time, the final effect created by "Lip-Flip" is amusing and entertaining for all-types of users.**

I. Introduction

The inspiration for this application comes from a sketch segment performed on Jimmy Fallon's *The Tonight Show.* In this segment, two separate cameras are directed at the faces of the host and his guest, and the lip region of each frame is swapped to the opposing camera. This creates an amusing affect where one user's mouth movements appear on the other user's face. However, the methods employed by the show are very rudimentary: the swapped portion of each camera is a fixed area, and both participants must keep their mouths centered on this region. Any small head movements immediately misaligns the lips. In addition, the minimal blending performed on the show is masked by the consistent, professional lighting used by the film crew. This is rarely feasible for standard users who might want to try "Lip-Flip."

The goal of this project was to create an entertaining, real-time, application which realistically swaps the mouths of two users using two webcams. We dynamically detected the location of each user's mouths and had the swapped lips follow the movement of the face. We also accounted for differences in image appearance between the two webcams, which would likely be the case when two users are using different computers. Like the sketch on *The Tonight Show*, we applied realistic blending so that each mouth was placed on the corresponding face in a believable manner. Our last, overarching goal was to insure that the application was fast, robust, and ran in real-time.

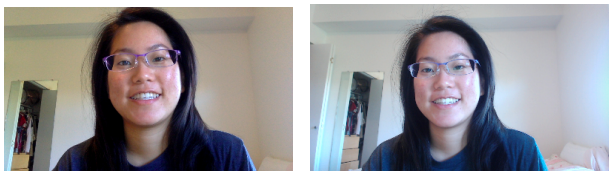
This application has the potential to be applied on video calling applications such as Google Chat or Skype. While users are talking with their friends, they can turn on "Lip Flip" and have their mouths swapped in real-time. Much like Google's "Google Effects," where virtual hats, glasses, and other accessories are dynamically inserted into a user's video stream, our Lip-Flip application has the potential to be an amusing accessory to video conference software.

II. Processing Pipeline

Our pipeline consists of the following steps. See Figure 1 for a graphical overview. On start-up, we perform a color and brightness calibration step. Here we find a transform that reduces the difference in appearance between images captured on each webcam. This transform is applied to all subsequent frames. For every frame, we use a Haar cascade face detector to find a region that contains a face. Next, we apply a nested mouth detector, focusing only on the bottom half of the face to locate the mouth. If no mouth is detected, an approximate region for the mouth is found using the detected face. With the known location of each mouth, we swap mouth regions between the two frames. By scaling each inserted mouth according to the region it is replacing, we account for the size of each face. Lastly, we apply Laplacian blending on each frame to blend the mouth onto the face. The processed frames are displayed on the screen, and the detection process is looped continuously to allow the application to run in real-time.

A. Camera Color Calibration

The first step in our pipeline is to transform our webcam images to have similar brightness and color. This is necessary because each user may have different webcams with different settings (e.g. auto-exposure, color balancing). If the differences between the two frames obtained from the webcams are not handled, the swapped lips will be particularly noticeable. Each lip segment will have a different appearance, even when the two subjects have the same skin color. An example of the differences between two webcams is shown in Figure 2.



(a) (b)

Fig. 2. Images from (a) Camera 1 and (b) Camera 2 may not have the same color or brightness

Our application accounts for this difference in the following manner. At the start of each session, a calibration step is performed. This calibration must be repeated with every new session, since the environment may change between different uses of the application. During calibration, the application takes two frames, one from each webcam. Next, it finds SURF keypoints in each image and calculates the closest matching keypoints. This step is sped up by using FLANN (Fast Library

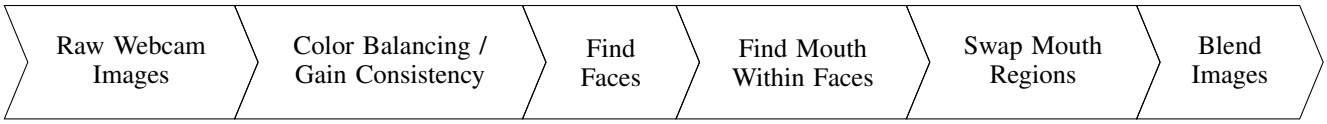


Fig. 1. Data Pipeline

for Approximate Nearest Neighbors) with OpenCV. Once matching keypoints are found, we estimate a homography using RANSAC, and keep only the inliers that fit this homography. This helps improve the validity of our matched pairs.

The inliers' coordinates give us a set of 2D image points that correspond to the same 3D world point. Ideally, we want the RGB values of these corresponding points to match between the two webcam images, so that the two frames look as similar as possible. Using these values, we find the best least-squares 4×4 transformation matrix that takes us from one set of values to the other. The inspiration for this calibration step came from [1].

$$\begin{bmatrix} R_{1,1} & G_{1,1} & B_{1,1} & 1 \\ R_{1,2} & G_{1,2} & B_{1,2} & 1 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} [4 \times 4] = \begin{bmatrix} R_{2,1} & G_{2,1} & B_{2,1} & 1 \\ R_{2,2} & G_{2,2} & B_{2,2} & 1 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

This calibration step is prone to finding poor transforms. First of all, not all correspondences found are always accurate. Secondly, the SURF keypoints tend to find dark regions that are not helpful for color calibration. Lastly, we occasionally do not find enough keypoints to solve for our unknowns. To mitigate these errors, we add a user-input step to verify the calibration. Upon start-up, the calibration continuously searches for matching keypoints until it finds enough to determine the best transform. When found, it applies the transform to the frame in question, and displays it for the user to see. If the transformed webcam image is significantly similar to the webcam we wish to match it with, the user will enter "yes" and the Lip-Flip session will start. If not, the program will continue looking for matching keypoints until it finds a transform that the user deems accurate. During calibration, the two webcams must have similar image views in order to ensure enough correspondences are found.

Once a single good transform is found, it is applied throughout the current Lip-Flip session. Every frame received from one webcam is transformed, according to the above equation, into a new frame that approximately matches the second webcam in terms of color, gain, and brightness. The results of one such transformation is shown in Figure 3.

B. Viola-Jones Object Detection

The next step in our pipeline is to detect the mouth region within each frame. We detect using the ViolaJones object detection framework [2] as implemented in OpenCV. Instead of working directly with image intensities, the Viola-Jones method uses Haar-like features to classify images. The original paper used features that consisted of two, three, or four rectangles. The value of a feature is computed by finding the difference in average pixel intensities between rectangles. In this manner, a single feature can help identify an object. For example, the three-rectangle feature shown in Figure 4

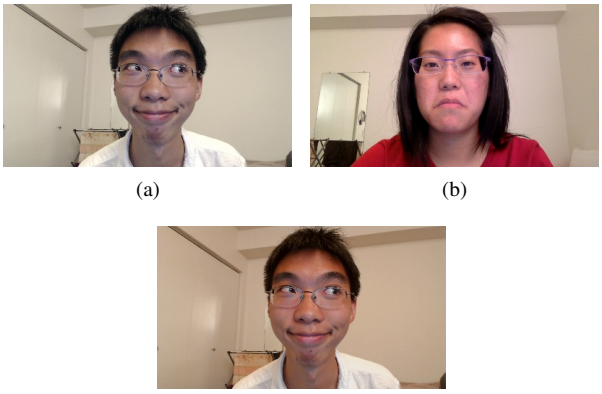


Fig. 3. Images showing how camera 1 (a) is calibrated to camera 2 (b) resulting in the transformed image (c)

is useful for detecting faces, due to the difference in pixel intensities between the eyes and the skin. In order to rapidly increase the processing speed of this algorithm, an integral image is used to quickly calculate feature values.

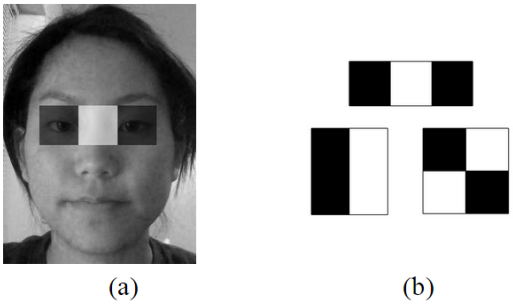
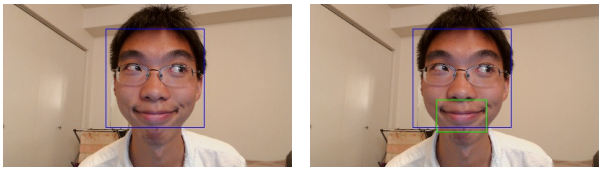


Fig. 4. (a) Example of a feature used for face detection. (b) Examples of Haar-like features used in the original Viola-Jones method.

A trained Haar cascade consists of various weak classifiers that have been arranged into a linear tree-like structure. A weak classifier is a Haar-like feature that can only correctly identify a positive training examples more than 51% of the time. The detection slides a window over an image at various scales and finds faces according to this cascade. The stages of the cascade begin with simple classifiers that can efficiently reject most windows and gradually increase in complexity. Early stages efficiently reject most window, while further stages ensure a low false positive rate. The cascade can be thought of as a linear tree; once a window is rejected by one stage, it is no longer processed with subsequent stages.

OpenCV provides trained cascades for various facial features. In our pipeline, we first use a face cascade to locate faces in each frame. We filter the detected faces and keep only the largest one. Next, we apply a mouth detector within

the bottom half of the face region. We use nested cascades to reduce the false positive detections that appear if one uses a mouth detector on the entire image. If no mouth is found, we use the lower third of the face region as the "detected" mouth. When a mouth region is determined, we expand it to be slightly larger than the detected region in order to have more pixels to blend with in our next step. An example of a detected face and mouth region is shown in Figure 5



(a) (b)

Fig. 5. (a) A detected face and (b) a detected mouth within the face

We insert each mouth region at the location of the mouth it is replacing. Mouths must be detected in both webcams in order for the swap to happen. Lastly, we scale each mouth region to match the width of the opposite mouth region. This allows the mouth to dynamically scale according to the size of the face in the image.

C. Laplacian Pyramid Blending

After finding the mouths in the two faces, scaling and swapping them is a fairly easy task. However, this produces an unrealistic image with a clear separation between the inserted mouth and original image. To reduce this problem and improve realism, we apply a blending algorithm to smooth out the transition.

For our Lip-Flip application, we decided to use Laplacian pyramid blending to blend the cropped mouth and the face. The Laplacian pyramid blending algorithm for two images A and B over a mask M works as follows:

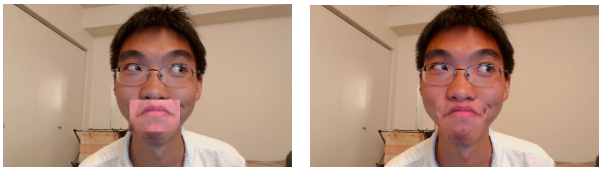
1) Generate a scale-space representations GA and GB for images A and B and GM for the mask M. To prevent aliasing from the subsampling, the image is Gaussian filtered before subsampling to generate the scale-space representation. Because the scale-space representation can be visualized as a "pyramid" of subsampled images, this is often referred to as a "Gaussian pyramid"
2) For the Gaussian pyramids GA and GB, generate Laplacian pyramids LA and LB. This is done by taking the difference between each scale representation and the following scale representation and generating a Laplacian for every scale
3) Combine the two Laplacian pyramids LA and LB into a combined pyramid LC by using the values of GM as weights. For each scale,

$$LC(i,j) = GM(i,j)LA(i,j)+(1-GM(i,j))LB(i,j)$$

4) To get the final combined image C we collapse the combined Laplacian pyramid LC by upsampling the image at each scale and adding them together

In our application, we take image A to be an image with a crude copy-paste of the new mouth scaled and placed over the destination mouth. Image B is the original image without a mouth copied over. The mask M was selected to be an ellipse covering the mouth region. Note that because we wish the blending to occur over a region around the mouth, the section of the mouth that is copied over must be slightly larger than the intended mouth region.

While we considered using a more complex and seamless blending algorithm (eg. Poisson blending) it could not perform fast enough for our real time application. The Laplacian pyramid blending algorithm satisfied this constraint, and as can be seen in Figure 6, does a very good job at blending the mouth on top of the original image.



(a) (b)

Fig. 6. Comparison of the mouth swap (a) without blending and (b) with Laplacian pyramid blending

III. Results

Our final application achieved the goals detailed in the beginning of this paper. An example of a frame being processed step-by-step is shown in Figure 7. The lips were swapped, dynamically tracked and scaled, and blended in a realistic manner. We achieved frame rates of around 15 fps for image sizes of 400x255 on a Macbook Air (13-inch, Early 2014).

IV. Future Work

Because the Haar-like features are not rotation or perspective invariant, detection often fails if a user rotates his or her head or looks away from the camera. Even if the detection succeeds, the cropped mouth is pasted at a visually awkward angle compared to the perspective of the face. Exploring different detection methods that are rotation invariant would improve the robustness of our app.

Another potential improvement would be to apply a method to take into account wide-open mouths. Currently, this case often leads to a failure in the mouth detection, since the Haar-cascades were not trained on open mouths. One option would be to train our own cascade, and when detecting an open mouth, apply some strategic changes to make the swapped mouths seem more natural. We did attempt to detect open mouths by sampling and comparing pixel intensities, but discovered the appearance of beards or shadows made the detection unreliable.

Another improvement we attempted was to improve blending for participants with differing skin colors. Currently, color balancing is performed on the entire image during our camera calibration step. We attempted to apply the same color/brightness matching on patches of skin in the swapped lip regions, but discovered that the differing color of the actual lips and skin produced unnatural results. These additional color balancing steps also dropped our frame rate. With different algorithms, such as fast lip segmentation, it may be possible to improve the blending between users of different skin tones.

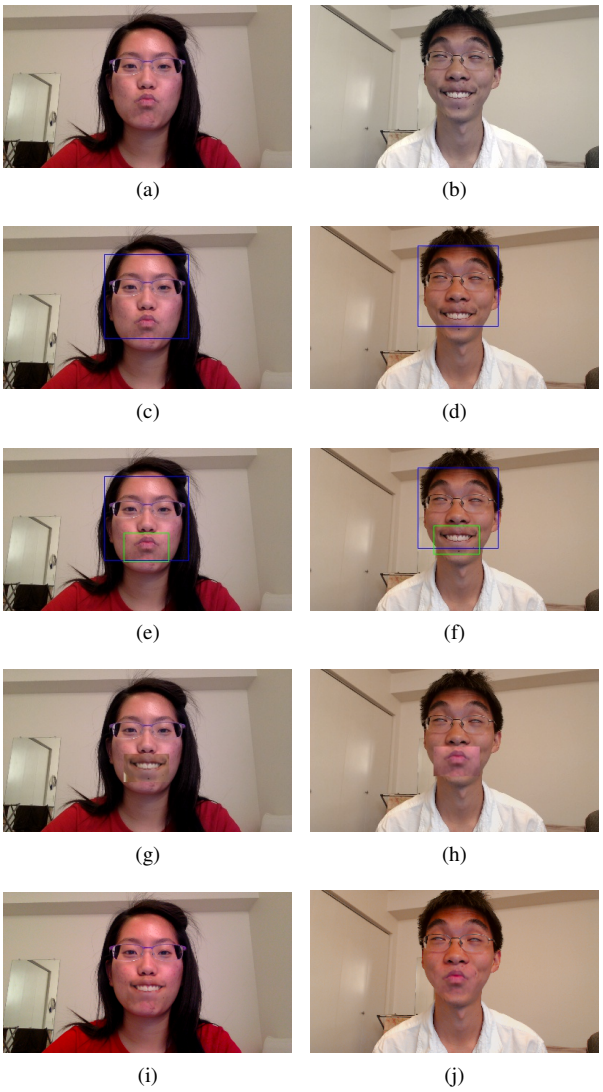


Fig. 7. Results from final application. From the raw images (a)(b), faces are detected (c)(d), mouths are detected (e)(f) and swapped (g)(h), then blended (i)(j) for the final lip-flip

V. Conclusion

We managed to create a very successful Lip-Flip application. Under typical conditions, the application reliably and efficiently detects the mouths and does a believable job swapping and blending the mouths. During our live demo, we received positive feedback on the entertaining and amusing aspects of our application. It was apparent that Lip-Flip could be an excellent add-on to a video conferencing program and provide amusement to users of all ages.

Acknowledgements

We would like to thank Professors Bernd Girod and Gordon Wetzstein as well as the TAs Jean-Baptiste Boin and Huizhong Chen for doing an excellent job running EE368 and making it a very enjoyable quarter.

We also want to give thanks to Roland Angst for providing the inspiration behind this project.

Appendix

A. *Work Allocation*

Trisha Lian - Camera calibration, Haar-cascade detectors

Kyle Chiang - Laplacian blending

References

[1] Xu, Ning, and James Crenshaw. "Image color correction via feature matching and RANSAC." Consumer Electronics (ICCE), 2014 IEEE International Conference on. IEEE, 2014.

[2] Viola, Paul, and Michael J. Jones. "Robust real-time face detection." International journal of computer vision 57.2 (2004): 137-154.

[3] Adelson, Edward H., et al. "Pyramid methods in image processing." RCA engineer 29.6 (1984): 33-41.