Real-Time Face Tracking and Replacement

Qi Cao
Department of EE
Stanford University
Stanford, CA 94305
qcao@stanford.edu

Ruishan Liu
Department of EE
Stanford University
Stanford, CA 94305
rliu2@stanford.edu

Abstract

In this paper we present an application on real-time face tracking and replacement. We implement this application through three sections: detection, tracking and replacement. For detection, we use haar features based cascade classifiers to detect the face, eyes and nose on both target and source images. Then face tracking is implemented by two distinct methods: CAMShift and Optical Flow. In cases users choose replacement, we cover the target face using a cartoon mask or blend it with a star face using Laplacian Pyramid Blending. To reduce the difference of skin colors between faces, we modify the Laplacian Pyramid Blending to better blend the target face with the source. The experiment shows that our methods lead to satisfactory tracking and replacement result, and our application works efficiently on mobile devices.

1 Introduction

Real-time video processing on mobile devices is a hot topic and has extensive applications. With a high-resolution display and high-performance camera, a modern mobile device can be a good platform for computer vision applications such as landmark detection, motion recognition, face tracking, etc.

This paper presents a mobile application combining face tracking and replacement in real-time videos. Capturing frames in real time via the camera of a mobile device, our application detects and tracks the face shown on the frames, which is called target face, and provides the users with the option to replace it with a source face. For replacement, users have two choices: one is to cover the target face with a cartoon mask; the other one is to blend it with a star face to change its facial features.

One of our motivations to create this application is privacy protection. Because of convenience, video recording and video calls via mobile device have a huge user group. Sometimes users may want to show something rather than their faces on the real-time video. A common instance is real-time distance teaching. Instructors may want to show slides or dancing movements without their faces appeared. However, most real-time video applications cannot meet such requirement. The alternative method is to manually process the video, which is time-consuming and lost its timeliness as well. Therefore, it is essential to develop an effective and real-time automatic face replacement method that can be implemented in mobile application.

Another motivation is for entertainment purpose. For example, when we watch a fantastic action movie, we may want to replace an actor's face with our own, to create our movie clip. Or in turn, we may want to replace our eyes an nose with a super star's when having a video call with friends, blending the "new face" with our own body and the real background to eliminate the artiface.

The structure of this paper is as following: In Section 2 we introduce related work and compare our methods with them. In Section 3 we give an overview of our application, briefly describing

its components and corresponding implementation methods. Sections 4 introduces face detection using haar features based cascade classifiers. In Sections 5 we detail two methods on face tracking: Optical Flow algorithm and Continuously Adaptive Mean Shift (CAMShift) algorithm. In Sections 6 we expatiate our methods on face replacement, including mask covering and modified Laplacian Pyramid Blending. Section 7 is dedicated to experiment on real-time videos captured via camera. Section 8 presents conclusions and possible future areas of work.

2 Previous Work

Face Detection: Existing strategies for face detection can be categorized in several groups, such as knowledge-based methods, feature invariant approaches, face template matching, and appearance-based methods[1]. For example, Hsu et al.[2] proposed a color feature based approach that searches face candidates based on the distribution of face colors in YCbCr space; Lanitis et al.[3] described a face by a set of shape model parameters and matches a query image with flexible appearance models.

In this paper, we apply the classical Haar-Like feature-based cascade classifiers[4] with some modifications introduced by Lienhart et al.[5]. The classifiers are pre-trained and can be loaded simply by CascadeClassifier class in OpenCV. Besides, compared with methods mentioned in last paragraph, the classifiers in our method are more robust and the computation is more efficient. For example, in [2], skin color varies with different human races and lighting conditions, but the Haar-like features we use are relatively invariant; cascade classifiers can discard negative images at early stages if one detection result of a weak classifier is negative. The method described in [3] is an optimization problem that requires many iterations, which are not efficient enough for mobile implementation.

Face Tracking: For face tracking, robustness to various target appearances and scene conditions are the main problems that need to be considered. Many methods have been discussed in current literature. For example, Schwerdt et al.[6] discussed a probability estimation method based on intensity normalized color histogram. It computes prior probability of skin color, and uses Bayes rule to get posterior probability of a pixel to decide whether it belongs to skin. Tresadern et al.[7] discussed an implementation of Active Appearance Model (AAM) that computes face shape and texture models in training, and fits them with the query image to locate the face.

We use two different methods–CAMshift[8][9] and Lucas-Kanade optical flow[10]–to track face, based on face detection result. Compared with previously discussed approaches, these two can be directly applied to current frame, without pre-training. Besides, they are both adaptive, based on the result of previous frame: CAMshift relocates and resizes search window at each frame, and optical flow method gets current keypoints and a homography transformation between previous and current frames.

Face Blending: The goal of face blending is to create a new face that combines features from original two or more faces. In existing literature, there is often a source face and a target face, and authors choose to put the eyes, nose and mouth of the target face on the source face. For static face blending, Poisson blending[11] is a commonly used strategy to reduce artifacts. [12][13] introduce a method for face blending in videos, which replaces a face in the target video by selecting the most similar candidate face from the source video.

The Poisson blending method can achive satisfactory results with almost no artifacts, but the runtime is not short enough for mobile implementation. On the other hand, the candidate selection strategy is efficient, but it requires a face library built from the whole source video. In our application, the source face is fixed (the given star face), so it is impossible to build a face library. For above reasons, we use Laplacian Pyramid blending[14] to blend the target face at each frame with the source face. This method is feasible, and can satisfy the efficiency requirement for mobile implementation.

3 Overview

Figure 1 shows an overview of our application. In order to replace the target face with a source face, we first detect the face on a real-time video using haar features based cascade classifiers. We use three pre-built classifiers (i.e., front face, big eyes and nose) to detect the face region and eyes and nose within that region respectively. After the user being satisfied with the detection result and giving us a command, we begin to track the face in each frame and replace it with a source face

chosen by the user (cartoon mask or star face) or not. We apply two independent methods on face tracking. One method is optical flow based tracking proposed by Lucas and Kanade. This method first extracts visual features from the region to be tracked, and then tracks it by estimating the optical flow between every two subsequent frames. The other method is CAMShift based tracking (Intel Corporation, 2001). This method is based on an adaptation of Mean Shift that, given a probability density image, finds the mean (mode) of the distribution by iterating in the direction of maximum increase in probability density. For replacement, our application provide users with three options: no replacement, covered by a cartoon mask, and blending with a star face. Since the CAMShift algorithm is a more robust technique, we apply it on the first two options. While for the blending option, we apply the optical flow algorithm because the homograph transformation matrix computed during tracking is useful to transform the source face for better blending. To blend the target face with a star face, we extract the eyes and nose regions of the star face, and align them to the target face on each fram. Then we blend them using modified Laplacian Pyramid Blending to create a new face.

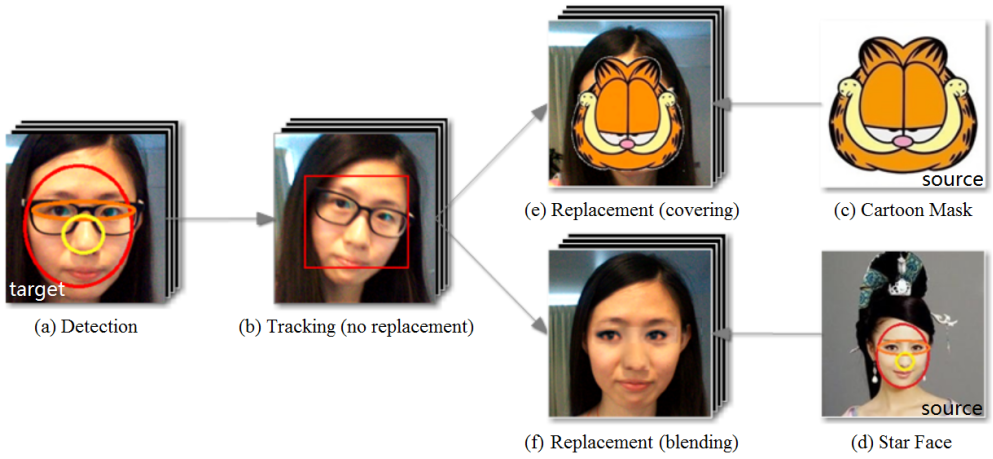


Figure 1: An overview of our application. (a) Face, eyes and nose detection result on each frame. (b) Face tracking using the detection result on the last frame in detection step (the red rectangle marks tracking region). (c) and (d) Soure face used for replacement. (e) Target face is covered by a cartoom mask on each frame. (f) Target face is blended with aligned eyes and nose of a star face on each frame

4 Face Detection

For our project, we assume only one face is contained in the video. The first procedure is to detect the face, with the eyes and nose within the face region, by Haar feature-based Cascade Classifiers.

Figure 2 shows some examples of Haar-Like features, which are represented as the sum of pixel intensities in black region minus that in white region. The algorithm first trains on a large number of positive(which contain faces) and negative(which do not contain faces) images, to select some relatively effective Haar-Like features.

For each feature, there is a corresponding weak classifier that outputs 1 for face, and 0 for nonface. Its performance is weak, that is, with high false positive or false negative rates. Thus, [1] proposes a boosting algorithm that boosts multiple weak classifiers into a strong classifier, and uses a sliding window to search the entire query image. At each stage, if the detection result of a weak classifier is below a certain threshold, the window is discarded immediately and will never be considered later. After passing all the stages, the remaining windows are considered to be face regions. The cascade classifier is shown in Figure 3.

OpenCV provides pre-trained classifier parameters (saved in XML files) for front face, eyes, and nose. Also, it provides CascadeClassifier class for these classifiers to detect at multiple scales. In our project, we load these classifiers as the detectors for face, eye and nose.

Before users choose "Face Tracking", we detect face, eyes and nose at each frame. First, we detect face and mark the face region; then, we detect eyes and nose within this region.

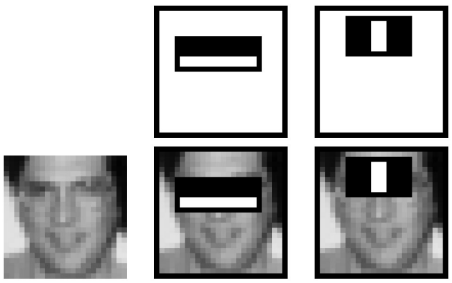


Figure 2: Haar-Like Features

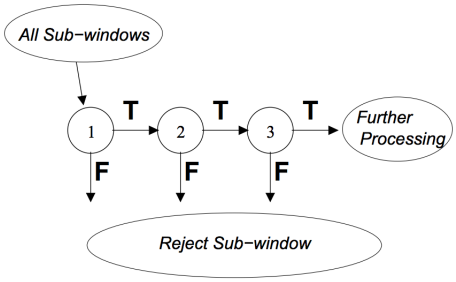


Figure 3: Cascade Classifier

5 Face Tracking

We apply two distinct methods on face tracking: Optical Flow algorithm and Continuously Adaptive Mean Shift (CAMShift) algorithm.

5.1 Optical Flow based Tracking

Optical flow is defined as spatio-temporal image brightness variations. To track a face across a sequence of frames, we use Lucas-Kanade optical flow algorithm[10] to estimate the motion of extracted facial features between adjacent frames in the sequence.

The algorithm has three assumptions:

1. Small motion. Points only have a small movement between two neighboring frames.
2. Brightness constancy. Projection of the same point in every frame has almost the same brightness.
3. Spatial coherence. Points should have almost the same movements as their neighbors.

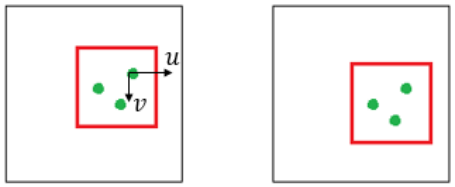


Figure 4: Points movement between two subsequent frames

The movement of a point can be described as a 2D vector $[u, v]^T$, as shown in Figure 4. Describe the brightness of point (x, y) at time t as $I(x, y, t)$, then, according to brightness constancy assumption,

$$I(x, y, t-1) = I(x + u(x, y), y + v(x, y), t) \tag{1}$$

Linearize it by Taylor expansion, we can get

$$\nabla I \cdot [u\ v]^T + I_t = 0 \tag{2}$$

Since the movements of neighbor points are almost the same, we can approximate the motion within a window of n points as

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_n) \end{bmatrix} \tag{3}$$

Then, we can estimate the positions of these points on the next frame as $p + [u\ v]^T$.

After we get the positions on the new frame, we estimate a 3×3 homography transformation matrix H between the two frames by RANSAC. Then, we transform the previous face bounding box, to get a quadrilateral on the new frame. For future face replacement, we adjust it to an upright rectangle by computing its minimum bounding rectangle, and set it as the new bounding box.

5.2 CAMShift based Tracking

The CAMShift algorithm[9] is an adaption of the Mean Shift algorithm for object tracking. The Mean Shift algorithm is a robust, non-parametric technique that climbs the gradient of a probability distribution to find the mode (peak) of the distribution.

The primary improvement of CAMShift is that CAMShift uses continuously adaptive probability distributions (i.e., distributions that may be recomputed for each frame) while Mean Shift is based on static distributions, which are not updated unless the object being tracked experiences significant changes in shape, size or color.

The CAMShift procedure can be summarized in Algorithm 1.

Algorithm 1 (CAMShift)

1: Set the region of interest (ROI) of the probability distribution image to the entire image.
2: Select an initial location of the Mean Shift search window. The selected location is the target distribution to be tracked.
3: Calculate the color probability distribution of the region centered at the Mean Shift search window.
4: Iterate Mean Shift algorithm to find the centroid of the probability image. Store the 0th moment (distribution area) and centroid location.
5: For the following frame, center the search window at the mean location found in step 4 and set the window size to a function of the 0th moment. Then go to Step 3.

The creation of the probability distribution function corresponds to steps 1 to 3. For initialization, we use the face region detected on the last frame in detection step as the initial location of the search window. Then, we need to calculate the color histogram corresponding to this region in HSV color space. Since human's skin colors have little difference in S and V channels, only the H (hue) channel is used to compute the color distribution, which consumes the lowest number of CPU cycles possible. The histogram is quantized into bins to reduce the computational and space complexity and allow similar color values to be clustered together. Then a histogram back-projection is applied.

Histogram Back-Projection: It is a primitive operation that associates the pixel values in the image with the value of the corresponding histogram bin. In all cases the histogram bin values are scaled to be within the discrete pixel range of the 2D probability distribution image using eq.(4).

$$\left\{ \hat{p_u} = \min\left(\frac{255}{\max(\hat{q})} \hat{q_u},\ 255 \right) \right\}_{u=1\ldots m} \tag{4}$$

Where $\hat{q_u}$ is the unweighted histogram corresponding to the n-th bin, and m is the number of bins.

That is, the histogram bin values are rescaled from $[0,\ \max(q)]$ to the range $[0, 255]$, where pixels with the highest probability of being in the sample histogram will map as visible intensities in the 2D histogram backprojection image.

Mass Center Calculation: The mean location (centroid) within the search window of the discrete probability image computed in Step 3 is found using moments. Given the intensity of the discrete

probability image $I(x, y)$ at (x, y) within the search window, the mass center is computed from:

$$M_{00} = \sum_x \sum_y I(x, y) \tag{5}$$

$$M_{10} = \sum_x \sum_y xI(x, y) \tag{6}$$

$$M_{01} = \sum_x \sum_y yI(x, y) \tag{7}$$

$$x_c = \frac{M_{10}}{M_{00}};\ y_c = \frac{M_{01}}{M_{00}} \tag{8}$$

Where M_{00} is the zero$^{\text{th}}$, M_{10} and M_{01} are the first moments for x and y respectively, (x_c, y_c) is the next center position of the tracking window.

The Mean Shift component of the algorithm is implemented by continually recomputing new values of (x_c, y_c) for the window position computed in the previousframe until there is no significant shift in position, i.e., convergence. Figure 5 shows one iteration of the Mean Shift algorithm.

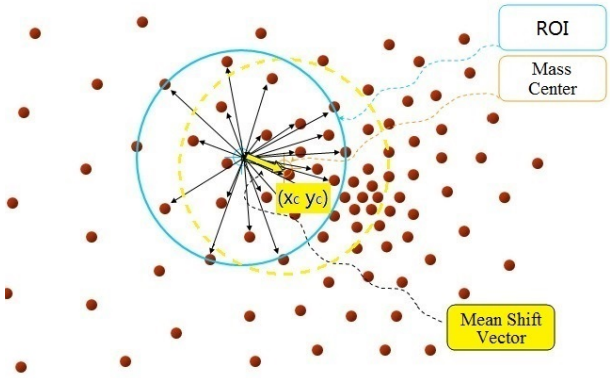


Figure 5: An Iteration of the Mean Shift Algorithm

6 Face Replacement

Our application provide users who want face replacement with two options: covering the target face by a cartoon mask or blending it with the eyes and nose extracted from a star face.

6.1 Covered by Cartoon Mask

Given an image of the face of a cartoon character. We turn it into a mask and use it to cover the target face in following steps:

1. Load the cartoon image and resize it to fit the tracking window (i.e., the box marking the target face region)
2. Check whether the tracking window is out of frame boundary. If so, set the region of interest (ROT) of the cartoon image to the one that corresponds to the face region within frame boundary. Otherwise, set the ROI to the entire image.
3. Make the white region (background) of the image transparent to form a mask.
4. Put the ROI of the mask onto the face region within frame boundary using Alpha Blending.

6.2 Blending with Star Face

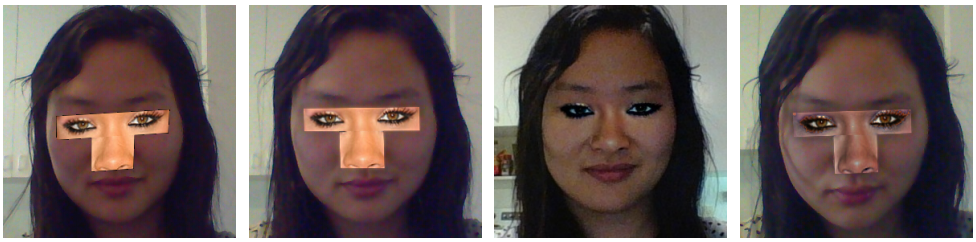
For this option, the target face is blended with the source face(star face). In specific, we replace the eyes and nose in the target image with those from the source image. However, face detection

at each frame is quite time-consuming, and the detection results for eyes and nose are both upright rectangles that cannot fully represent perspective transformations. For these reasons, we choose to detect the source image at first, to get its eyes and nose. Then, clone them to a new blank image I_0, resize and relocate them to fit the sizes and locations in target image. During tracking process, we warp I_0 to I_w by an overall homography H_o between the last detection frame and current frame, and blend the current frame with the warpped image I_w. H_o is set as I at the beginning, and is updated in tracking process by

$$H_o = HH_o \tag{9}$$

where H is the homography transformation matrix between adjacent frames.

There are multiple face blending methods: alpha blending, pyramid blending[14], Poisson blending[11], etc. As shown in Figure 6, alpha blending is the easiest one, but it cannot avoid the seams around blending contours; pyramid blending can solve the problems of seams, but there are discrepancies between skin colors of source face and target face; Poisson blending can almost avoid artifacts, but the runtime is long, about 30s for each frame, so it is not suitable for our app. Our strategy is to adjust the skin color of source face before blending, then blend the two faces by Laplacian Pyramid Blending.



(a) Alpha Blending (b) Pyramid Blending Without Color Adjustment (c) Poisson Blending (d) Modified Pyramid Blending

Figure 6: Comparision of Different Face Blending Methods

Skin Color Adjustment: After detecting the eyes and nose of source image, we need to reduce the differences of skin colors between source and target images. Since skin color of the star face on a source image is often very bright due to spotlight, we preprocess the image with gamma correction, with $\gamma = 1.5$.

We make an assumption that the skin colors of eyes and nose for a face are similar. First, we compute the means of nose colors in both source and target images, denoted as $\bar{x}_s$ and $\bar{x}_t$, respectively. So the difference is

$$d = \bar{x}_t - \bar{x}_s \tag{10}$$

For nose, we just add d to the nose region in source image, to adjust its color. For eyes, we get the regions of skin in source eyes, which does not include eyeballs and eyebrows; then add d to all the pixels in the skin region. Skin region is detected by a threshold above and below $\bar{x}_s$, that is, if the BGR color x of a pixel satisfies $\bar{x}_s - a_1 \leq x \leq \bar{x}_s + a_2$, it belongs to skin region, where a_1 and a_2 are two thresholds. The whole process of skin color adjustment is shown in Figure 7.

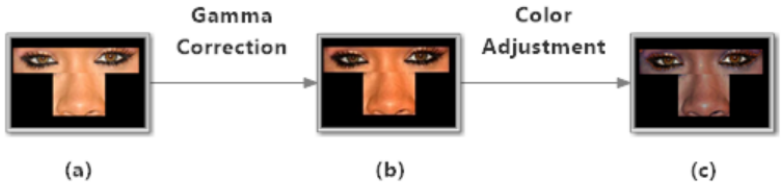


Figure 7: An Iteration of the Mean Shift Algorithm

Laplacian Pyramid Blending: After skin color adjustment, we can get the image I_0 that only contains eyes and nose from source image. Compute a mask M_0, in which the region of eyes and nose is 1, and other region is 0. At each frame, we warp I_0 and M_0 by H_o to get I_w and M_w, then blend target image I_t with I_w. The blending algorithm is summarized in Algorithm 2.

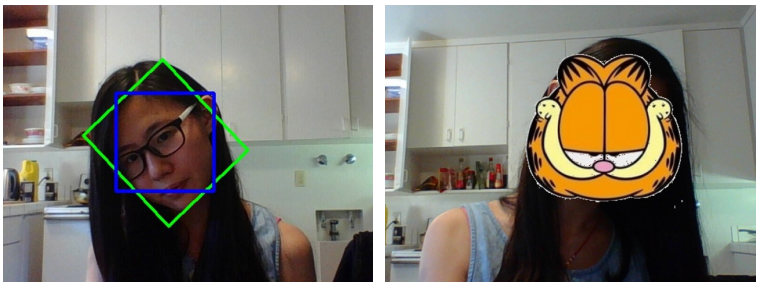
Algorithm 2 (Laplacian Pyramid Blending)

1: Build Laplacian pyramids L_w and L_t from I_w and I_t.
2: Build Gaussian pyramid G_m from mask M_w.
3: Form a combined pyramid L by $L = G_m * L_w + (1 - G_m) * L_t$.
4: Collapse the L pyramid to get the blended image.

7 Experiments

We test our application on a laptop PC and a tablet. On the laptop, we use both CAMShift and Optical Flow based methods on face tracking for all options, to evaluate performance and decide which method(s) to apply for each face replacement option, respectively. Then we implement them on both the laptop and the tablet to satisfy users' different requests, and compare the efficiency of our application on the two platforms. For the laptop, we use a Lenovo E440, with a 2.20 GHz Intel-core i7 CPU and a 4 GB RAM; for the tablet, we use a Nvidia Shield Tablet that has a Nvidia Tegra K1 mobile processor, a 192 core Kepler GPU and a 2 GB RAM. The resolutions of PC capture and tablet camera are both 640×480.

To choose the tracking method for each replacement option, we compare the performance of face tracking between the two methods: CAMShift and Optical Flow.



(a) No Replacement, CAMShift based Tracking

(b) Cartoon mask, CAMShift based Tracking

(c) No Replacement, Optical Flow based Tracking

(d) Cartoon mask, Optical Flow based Tracking

Figure 8: Comparision of Different Face Tracking Methods

As shown in Figure 8, CAMShift is more robust under large movement conditions, such as strong head rotation and fast head translation. Besides, it can recover the face even it is previously out of frame bound. Compared with CAMShift, Optical Flow method cannot effectively handle large movements, and is hard to relocalize face after lost. This is because in Optical Flow method, we have

made the small motion assumption, to ensure Eq.(2)(3) hold. For large movement, they cannot hold anymore, so there is large error in computing $[u\ v]^T$ from Eq.(3). Besides, if the head is strongly rotated, the spatial coherence assumption cannot be satisfied, since the rotations of one point and its neighbor are different. Thus, we choose CAMShift method if users choose face tracking or cartoon mask options.

Considering the fact that we need to align the source image with the target image before blending them, we need to apply a homography transformation on the source image. If using CAMShift method, we need to extract keypoints at each frame, then estimate homography matrix H. However, Optical Flow method already provides us with keypoints, so we only need to estimate homography. For computation efficiency, we choose Optical Flow method if users choose face blending option.

Our application on the tablet begins face detection after users tap the screen. The face, eyes and nose will be marked by ellipses of different colors. Then, users can tap the screen to begin face tracking, and can choose a replacement option. In tracking process, users can also tap the screen to restart face detection. Figure 9-12 shows the user interface of our application.

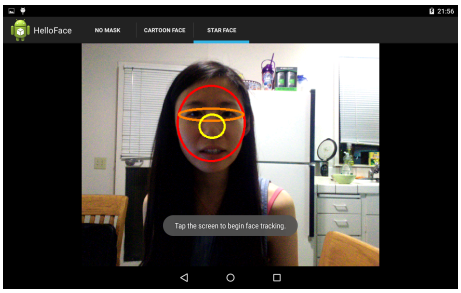


Figure 9: Face Detection Interface

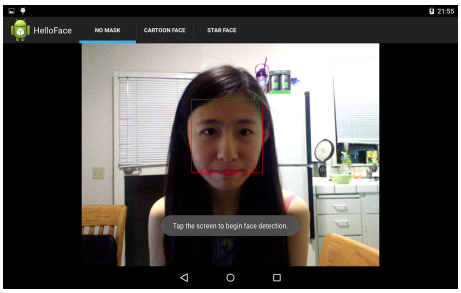


Figure 10: Face Detection Interface

Figure 11: Face Detection Interface

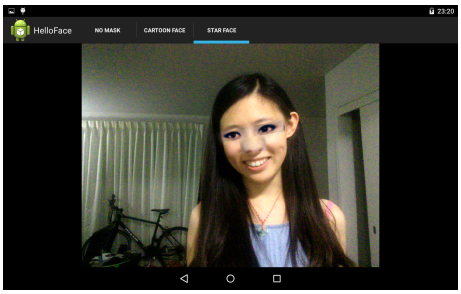


Figure 12: Face Blending Interface

To estimate the efficiency of our application, we calculate the average runtime per frame for each procedure. As shown in Table 1, although the time taken on the tablet is 2-4 times as much as the time taken on the laptop, it is efficient enough for real-time mobile implementation.

Table 1: Comparison of Performance for Each Procedure in Terms of Efficiency

Method	Time Taken (ms/frame)	
	Laptop	Tablet
Face Detection	52.387	117
Face Tracking (no replacement, CAMShift)	10.846	43
Face Replacement (cartoon mask, CAMShift) S	15.974	48
Face Replacement (blending, optical flow)	340.218	404

8 Conclusions

Our application can satisfy real-time face tracking and replacement requests on Android devices. For face detection, Haar-Like feature-based cascade classifier is an efficient, easy to implement method. For face tracking, CAMShift can handle large and fast movements, which is morSe robust than Optical Flow method. For face blending, Optical Flow is more efficient, since it provides necessary keypoints for image alignment.

Compared with other image blending methods, our method of skin color adjustment and pyramid blending can reduce skin color difference between the star face and user's face, and is efficient enough for mobile implementation.

To improve our application, future work will focus on better blending methods that can seamlessly blend two faces and reduce the runtime to be more suitable for mobile devices.

References

[1] Wenyi Zhao, Rama Chellappa, P Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM computing surveys (CSUR)*, 35(4):399–458, 2003.

[2] Rein-Lien Hsu, Mohamed Abdel-Mottaleb, and Anil K Jain. Face detection in color images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):696–706, 2002.

[3] Andreas Lanitis, Christopher J Taylor, and Timothy F Cootes. Automatic face identification system using flexible appearance models. *Image and vision computing*, 13(5):393–401, 1995.

[4] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[5] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *Pattern Recognition*, pages 297–304. Springer, 2003.

[6] Karl Schwerdt and James L Crowley. Robust face tracking using color. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 90–95. IEEE, 2000.

[7] Philip A Tresadern, Mircea C Ionita, and Timothy F Cootes. Real-time facial feature tracking on a mobile device. *International Journal of Computer Vision*, 96(3):280–289, 2012.

[8] Gary R Bradski. Computer vision face tracking for use in a perceptual user interface. 1998.

[9] John G Allen, Richard YD Xu, and Jesse S Jin. Object tracking using camshift algorithm and multiple quantized feature spaces. In *Proceedings of the Pan-Sydney area workshop on Visual information processing*, pages 3–7. Australian Computer Society, Inc., 2004.

[10] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[11] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 313–318. ACM, 2003.

[12] Dmitri Bitouk, Neeraj Kumar, Samreen Dhillon, Peter Belhumeur, and Shree K Nayar. Face swapping: automatically replacing faces in photographs. *ACM Transactions on Graphics (TOG)*, 27(3):39, 2008.

[13] Yu Liang, Bing-Yu Chen, Yung-Yu Chuang, and Ming Ouhyoung. Image based face replacement in video. *Master's Thesis, CSEI Department, National Taiwan University*, 2009.

[14] Edward H Adelson, Charles H Anderson, James R Bergen, Peter J Burt, and Joan M Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.