Face Emoticonizer: Facial Feature Detection and Changing on Android

Kun Yi (kunyi AT stanford.edu)
Department of Electrical Engineering
Stanford University

***Abstract*—With the aim of adding vividness to video chat, we implemented an Android application that puts user-selected emoticons on human faces in real time. The Viola-Jones object detector for detecting faces and facial features, and two algorithms are proposed to correct facial feature positions and remove features. Though being simple, we conclude the application achieves its job effectively.**

I. Introduction

An emoticon, or 'kaomoji' in Japanese romanji, stands for a pictorial representation of a facial expression using characters, and is often used for adding an emotional context in text communications. We are motivated by the popularity of emoticons, and think it would be interesting to add the comic favor of the emoticons to the real human face. This paper introduces Emoticonizer, an Android mobile phone application developed that can track the facial features and replace them by user-input emoticons in real time. Such a functionality could potentially be fun and useful for applications such as video chatting, photo editing, and gaming.

The rest of the paper is organized as follows: section II introduces related work in the area of facial feature detection and morphing. Section III describes our algorithm for facial feature replacing. In section IV, we demonstrate working examples as well as experimental results using the application. Finally, comments about the design trade-offs we made and potential improvements are made in section V.

II. Related Work

A. Face and Facial Feature Detection

Face detection and facial feature detection /tracking has received extensive study in academia, and potential applications cover a wide range such as emotion extraction [1], and video chatting [2]. Broadly, the detection techniques can be training-based or blind, and a face geometric model is often taken into consideration for reference of the positions of the features.

One particular face/facial feature detector, Viola-Jones object detector [3], is a training based detector that uses Haar-like features for image classification. Briefly, Haar-like features can encode the orientation information between regions in the image. Human face and facial features can be encoded by a set of such features, and also several detection stages can be integrated to form a "cascade classifier" in order to improve performance. Because each feature is one simple small image, the detection can be done extremely fast, however trading off robustness for speed.

B. Blending

Blending or smoothing is required as we remove the facial features and draw the characters in order to improve visual quality of the composed image. A simple technique is alpha-blending, where the images are combined with the overlapping area being a linear combination of the borders. Gaussian or Laplacian Pyramids [4] and Poisson image editing [5] are examples of proposed techniques with better effect, yet at the expense of increased computational complexity due to gradient calculations or least-square solutions.

III. Algorithm

Figure 1 shows the sample work flow of Emoticonizer. We describe the four major steps here: facial feature detection, position correction, feature removal, and drawing emoticon.

A. Facial Feature Detection

The first step involved is face detection. Once a video frame is received from the Android phone

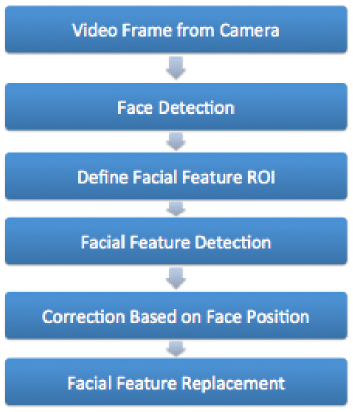
Video Frame from Camera
Face Detection
Define Facial Feature ROI
Facial Feature Detection
Correction Based on Face Position
Facial Feature Replacement

Fig. 1. Application Workflow

camera, we run Viola-Jones face detection on the frame, with a trained Haar-like cascade classifier for face provided by the OpenCV library [6]. Output obtained consists of the center position, width, and height of the detected face object(s). Based on the position and size information of the face region, we then define one Regions of Interest (ROI) for each facial feature in order to restrict the search for features within reasonable areas.

Finally, we run Viola-Jones on each ROI defined, this time with the trained classifiers for facial features. In our particular case, we are only detecting the eyes and the mouth.

B. Facial Feature Position Correction

One challenge in robust feature detection imposed by real-time video processing is the movement of the camera and the objects. As discussed before, the Viola-Jones detector is not perfectly robust, and we found such movements can easily cause it to fail recognize the facial features once in a few frames. Therefore, naive method where we simply run the detection from frame to frame will have the problem of losing features.

We propose a simple tracking algorithm to improve the robustness of feature detection: correction based on face position. Specifically, we found that the detection of face is often more stable than the detection of features, therefore when features are not detected or detected with wrong positions in a frame, we can use the offset of the face position from a previous frame to correct/calibrate the position. The simple correction technique is described in the following steps:

Algorithm 1 Facial Feature Position Correction

```
shouldCorrect = false
if feature.center = NULL or feature.center not in ROI then
    shouldCorrect = true
    offset = face.center - facePrev.center
    feature.center = feature.center + offset
end if
```

C. Feature Removal

In order to get rid of the original facial features before we draw the emoticon characters, we need to ”remove” those features by filling in skin color. We first sample a specific area on the face region, and obtain the histogram of R,G, and B values. Based on the these trained values, a ”reference skin color” can be calculated as the average. However, we try to avoid simply filling in the whole rectangular region detected with a fixed skin color, as the sharp contrast at the edges will greatly reduce image quality.

In order to improve visual effect, we propose the following blending technique. It is a variation of the alpha-blending with the foreground being the feature image and the background being the skin color image. However, linear combination is done only when the color of the pixel is between a certain threshold within the sampled skin color. We define a simplified color distance function

$$\begin{aligned} &dist([R_1, G_1, B_1], [R_2, G_2, B_2]) \\ &= max(|R_1 - R_2|, |G_1 - G2|, |B_1 - B2|) \end{aligned}$$

Our algorithm for filling skin color is presented in Algorithm 2. Also it can be visualized in Figure 2

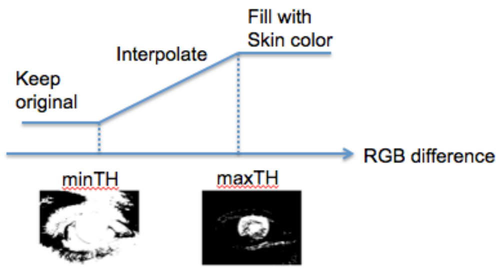

Fig. 2. Blending Algorithm. Left mask shows pixels with > minTH to skin color, right mask shows pixels with > maxTH to skin color. Everything in between will a linear interpolation of original and skin color.

Algorithm 2 Feature Removal

```
for row in region do
    rowSkin = number of pixels with dist(pixel.color, skinColor) $>$ minTH
    ratioSkin = rowSkin/row
    if ratioSkin $>$ 0.9 then
        break
    else
        for pixel in row do
            diff = dist(pixel.Color, skinColor)
            if diff $<$ minTH then
                break
            else if diff $>$ maxTH then
                pixel.Color = skinColor
            else
                alpha = diff/(maxTH - minTH)
                pixel.Color = pixel.Color * (1-alpha) + skinColor * alpha
            end if
        end for
    end if
end for
```

The rationale is that: any pixel with value reasonably close to skin color could be due to shading, lighting, etc., and we would want to keep a proportion of the original color to smooth the image. Once the process finishes, we apply a Gaussian blur filter to further smooth out the transition between. In practice, we find the values $[minTH, maxTH] = [30, 120]$ results in good performance.

Figure 3 demonstrates a Matlab generated example of the algorithm. As we can see, the blending almost removes all the non-skin color pixels, but preserves the shading around the eye.

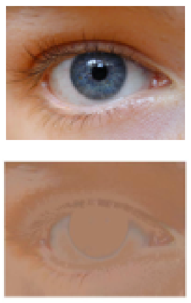

Fig. 3. Top: original image. Bottom: blended image

D. Drawing Emoticon

Once we remove the facial features, it is quite easy to draw the emoticons on top. A number of emoticons have been pre-stored in the phone, and already broken up to sub-images of eyes and mouth. We simply scale each subimage to the size of the corresponding detected facial feature, and draws on top of the already smoothed face. After this step, the processing of the image or video frame is done, and is sent to the phone for display.

IV. Experimental Results

The application is tested on an Moto Atrix phone, with dual core 1GHz processors, 2GB of memory, and a 5MP camera, running an Android 2.3.6 system.

A. Image Quality

We tested the algorithm on a variety of face inputs, including photos displayed on a computer screen as well as real human faces. The contrast and lighting conditions also vary in these situations. Figure 4 and Figure 5 demonstrates some working examples. The first is using camera to capture a high-resolution standard photo displayed on the computer screen, and the second is obtained by using camera to capture human face directly.

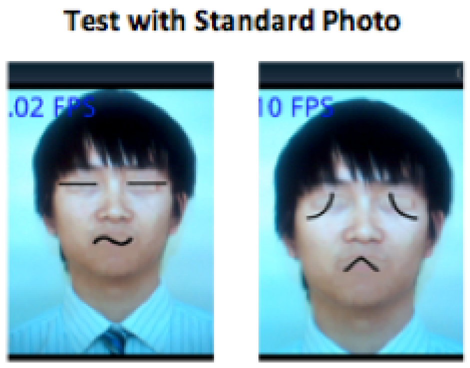

Fig. 4. Algorithm tested with photo inputs.

Although image quality can be a subjective metric, we can still clearly see that the blending quality using photo is better than the real video camera input case. What is more, with mouth open, the position of the mouth is detected not accurately enough. The blending doesn't work as good on video side too. The mouth section in the lower right can still be distinguished from the surrounding area, leading to undesirable visual quality.

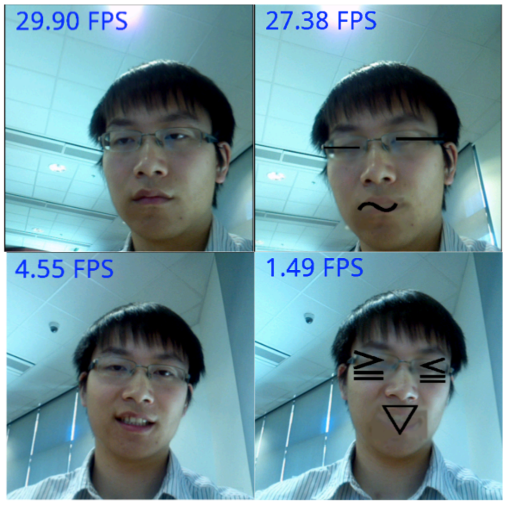

Fig. 5. Algorithm tested with face input.

Figure 6 exemplifies the problem with sampling skin color from a fixed region. As we can see, since the reference skin color differs too much from the rest of the face, blocking effect appears at eyes and mouth. To address this problem, local sampling must be taken into consideration.

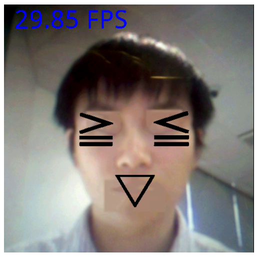

Fig. 6. Algorithm tested with face input with a different lighting condition.

B. Running Time and Stability

Our application runs satisfactorily in real-time, with an average speed of as high as 30 fps and as low as 5 fps when tracking the facial features for one face. At this speed, the lag in drawing a new frame when the face moves is tolerable. We this the running speed could be improved by implementing our position correction algorithm and feature removal algorithm in native OpenCV C++ code, which runs faster than Java.

The application runs stably in normal conditions. However, we notice that often when experiencing rapid lighting condition changes or detecting a face with high contrast, the running speed will drop to below 2 fps and often causing the application to crash. We suspect this is due to some corner cases that caused a bug in our code.

V. Future Work and Conclusion

There are several aspects that can be potentially improved:

- The face tracking algorithm. Currently our algorithm simply uses the face position to correct feature detections. This tracking algorithm can be improved in two ways. Firstly, an improved geometric model of the face and features, such as the one mentioned in [2], can be used. Second, the face tracking can be done using tracking algorithms such as Camshift [7]. These tracking algorithm often utilize the difference between frames to track, so computational complexity will be reduced since we don't need to detect face for each frame.
- Blending algorithm. If we can optimize the implementation, it might be possible to adopt a more complex algorithm for blending, so as to improve the effect.
- Simple emotion detection. It would be much more interesting if the emoticon can be automatically generated based on the emotion detected. Although emotion extraction in general is a hard problem, we think there is room for simple detection. For example, based on skin color classification, we can detect blinking by detecting whether eye object is present in the ROI, and let the emoticon blink at the same time. Such tricks could potentially make the emoticon presentation more vivid.

In conclusion, we have achieved our goal by building a fun application that can make emoticons out of human faces. Although facing limited computational power particular to the mobile phone platform, our application runs with satisfactory speed. I have learned great lessons as well as had much fun from implementing the application, and the experience would be invaluable for sure.

VI. Acknowledgement

The authors would like to thank Professor Bernd Girod, teaching assistants David Chen and Derek Pang, and mentor Huizhong Chen for all the advice they provided throughout the project.

References

[1] K.-E. Ko and K.-B. Sim, "Development of the facial feature extraction and emotion recognition method based on asm and bayesian network," in *Fuzzy Systems, 2009. FUZZ-IEEE 2009. IEEE International Conference on*, aug. 2009, pp. 2063 –2066.

[2] Z. Zhu and Q. Ji, "Robust pose invariant facial feature detection and tracking in real-time," in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, vol. 1, 0-0 2006, pp. 1092 –1095.

[3] P. Viola and M. 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*, vol. 1, 2001, pp. I–511 – I–518 vol.1.

[4] P. Burt and E. Adelson, "The laplacian pyramid as a compact image code," *Communications, IEEE Transactions on*, vol. 31, no. 4, pp. 532 – 540, apr 1983.

[5] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 313–318, Jul. 2003. [Online]. Available: http://doi.acm.org/10.1145/882262.882269

[6] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[7] G. R. Bradski, "Computer vision face tracking for use in a perceptual user interface," 1998.