# CS231M Project Report - Automated Real-Time Face Tracking and Blending

Steven Lee, slee2010@stanford.edu

June 6, 2015

## 1 Introduction

Summary statement: The goal of this project is to create an Android application that automatically detects, locates, and swaps faces in real-time during video capture.

### 1.1 Motivation

Facial recognition is an important task in a wide variety of industries such as security, marketing, and entertainment. In many security applications, users often need to detect and locate faces in real-time in surveillance footage. Similarly, marketing applications such as smart billboards need to detect when people are facing them in order to serve dynamic content. Additionally, applications often use facial recognition in order to deduce a person's identity from an image of their face.

In the entertainment industry, face-swapping can be a useful tool for Computer-generated imagery (CGI) in television or film. For example, films often want to superimpose an actor's face onto multiple bodies for purposes of duplication or stunt double consistency (See Figure 1).

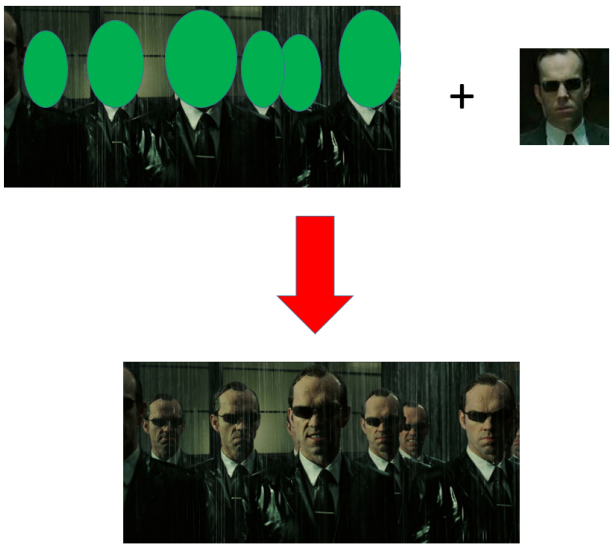


Figure 1: Example schematic for face-swapping usage in entertainment. Image source - The Matrix Revolutions.

However, this application will primarily contain entertainment value, as face-swapping is a fun way to make goofy images with friends.

### 1.2 Problem Description

The major technical challenges for this project are detecting, locating, and swapping faces in real-time. To simplify the problem, the application should first prompt the user for a static "source" face which will replace all detected faces in subsequent video. Detecting faces poses the challenge of localizing faces while being robust to varying lighting, angle, pose, and size. Swapping faces poses the challenge of blending or stitching an image into a space that may not match the face's original size. Finally, running this in real-time requires efficiency measures - for example, we can utilize flow tracking to avoid re-detecting faces in each frame. To summarize, the tasks required for this application can be divided into three primary goals:

1. Face Localization. The input to this system is a video stream with zero or more faces per frame. This system must output the same video stream, but for every frame, must augment the frame with bounding boxes around each detected face.

2. Face Blending. Given a frame and bounding boxes around each face, this system must blend each detected face with some face supplied by the device or user.

3. Real-time performance. The goal is for this application to run in real time (roughly 30 frames per second) on a mobile device. For this project, the target device is the NVIDIA Shield tablet.

## 2 Previous Work

### 2.1 Review of Previous Work

Previous work has often used facial detection in industries with financial incentive. For example, facial detection is an important tool in marketing, where a business might want to track the number of people that enter a shop on a given day. Using a webcam that has a continuous video feed of a store entrance, a system can count the number of customers that enter the store and try to approximate their ethnicities, genders, or age based on their facial features. One such system is the OptimEyes facial tracking and human face classification system that is integrated into a billboard product called Amscreen [1].

Face detection and localization is also used ubiquitously in current-generation digital camera software and image sharing. When most modern digital cameras obtain a live view through their lenses, they usually try to autofocus the image. To determine which regions of interest to focus the image, camera systems can use facial detection and localization.

While we use Haar features with a cascade of classifiers for facial detection, previous methods have used different approaches to the similar problem. Early systems used feature matching to detect faces. For instance, a system would try to locate specific features of a face - such as a jaw, nosebridge, or eyebrows - and use their relative positions to detect and locate a face. Principal component analysis has also been used to train systems to train and construct eigenfaces, which are then used rank regions of interests with a probability of containing a face. Additionally, previous algorithms have used normalization on a library of faces to obtain a compressed "average" face, from which they match potential regions of interest.

### 2.2 Main Contributions of This Work

In this project, we use a system of Haar features with a cascade of classifiers to detect and locate faces in a live video feed. Along with facial blending, this project's main contributions are:

1. Exploration of real-time performance of the Viola-Jones tracking system on a mobile device [2]. Previous work has often used servers or desktop platforms to compute face detection and localization. A mobile platform has less memory, electrical power, and computational power.

2. Pairing of a facial detection and localization system with a face blending system for an entertaining mobile application.

3. Enabling facial detection and localization in a portable, lightweight system, because Android mobile devices are widely used today.

# 3 Technical Method

## 3.1 Technical Summary

Face detection and localization is achieved with two approaches:

1. Independent face detection and localization in each frame.
2. Face detection and localization in the initial frame, followed by an independent KLT and ORB plane tracker for each face found.

Face blending is achieved with alpha blending, as Poisson blending was found to be too computationally intensive for a mobile platform.

## 3.2 Technical Detail

### 3.2.1 Target Mobile Platform

The target mobile platform is the NVIDIA Shield tablet, with relevant hardware specifications below:

1. 2592  1944 pixel rear-facing camera.
2. 2592 x 1944 pixel front-facing camera.
3. Quad-core 2.2 GHz CPU.
4. ULP GeForce Kepler (192 processing units) GPU.
5. 2 GB memory.
6. Android Lollipop (API 21) operating system.

### 3.2.2 Computing Pipeline

The flow of image data for this application is as follows:

1. An image frame captured from the device's camera is passed into the Android application.
2. For the first image frame, the application initializes all necessary data structures and tracking systems required for face detection and tracking for future frames.
3. For all subsequent image frames, the application locates all faces, or determines that they have moved out of frame.
4. For each detected face, the application blends it with a face supplied from a saved file.
5. The image frame is augmented with a bounding box and blended face.
6. The image is frame is immediately presented to the user application.

### 3.2.3 Face Detection and Localization

Face detection uses the Viola-Jones face tracking system. At a high level, for a given image frame, the system searches many regions of interest in the frame. For each region of interest, the system computes a set of Haar features on the frame, and uses a cascade of classifiers to determine whether or not to reject this region of interest (i.e. determine that it contains no faces) or determine that it does contain a face [**?**].

A Haar feature is a way to compress spacial information from a region of interest into a scalar value. In Figure 2, we observe that a Haar feature can be expressed as a binary image mask. To compute the Haar feature for a given region of interest, we compute:

$$\sum_{p \in W} p - \sum_{p \in B} p$$

Where $W$ and $B$ are the set of pixels in the region of interest under a white pixel or black pixel in the Haar feature mask, respectively. For a given region of interest and Haar feature, this gives us a scalar value to use as input for classification. For this project, we use a pre-trained set of Haar features from Viola-Jones that are optimized for facial detection.

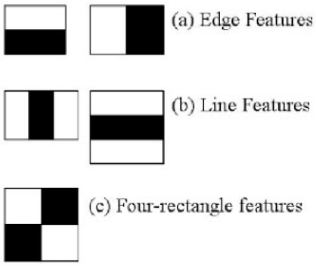



Figure 2: Example of Haar feature masks, which are used to compute scalar feature values for a region of interest.

To determine whether a region of interest contains a face or not, we use a computationally efficient system called a cascade of classifiers. Because computing the entire feature set for every candidate region of interest would be computationally very expensive (even for a machine with more resources than our target mobile platform), we would like to reject a region of interest early, before computing all features.

The cascade of classifier works as follows: for a given region of interest, we repeatedly apply a stage (i.e. a small subset) of our Haar features. Using these feature values and their corresponding weights pre-trained by the classifier, we compute a score for the region of interest. If the score is below a certain threshold, we can immediately reject this region of interest and avoid computing more features for it. If the score is at least the threshold, we continue to apply more stages on this region of interest. If a region of interest passes all stages, we determine that it contains a face.

The stage sizes increase as they progress (e.g. the one used in this project goes from 1, 10, 25, 25, 50 features in the first five stages). The intuition behind this is that we would like to reject the low-hanging fruit (i.e. regions of interest that can be easily eliminated using one Haar feature) before spending more computational resources calculating additional Haar features. As a region of interest passes more stages, we apply a stricter (i.e. larger) set of features so we can be confident that resulting regions of interest contain faces.

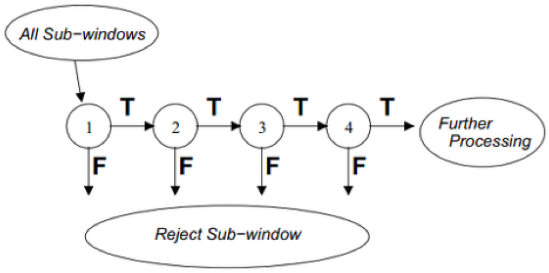



Figure 3: General schematic of cascade of classifiers, during which regions of interests are iteratively evaluated with a stricter set of Haar features to determine the presence of a face.

#### 3.2.4 Face Tracking

Face tracking is achieved using two methods:

1. Independent Viola-Jones face detection and localization in each frame. This system implements the Haar feature calculation and cascade of classifiers for each frame, independent of the previous or future

frames. The system does not maintain state about any located face, since faces are detected only for a given frame as it is given to the system.

2. Viola-Jones face detection and localization for initialization, followed by a KLT and ORB plane tracker for each face found in the initial frame. This system uses the Haar feature calculation and cascade of classifiers only for the *initial* frame, from which it obtains a set of faces. For each face in this set, the system initializes a KLT and ORB plane tracker with reference bounding box equal to the bounding box of the face found in the frame. Using features calculated in the initial bounding box, this plane tracker is able to track the movement of this plane in 3-dimensional space as the plane undergoes projective transformations. For each subsequent frame in which the plane tracker finds a face, the system estimates a homography for the transformation from the face's original bounding box to its current bounding box. The system can then use this homography to recalculate the new bounding box for the face. The plane tracker does have limitations, including the assumption of planarity of tracked faces. This assumption is weak because faces are 3-dimensional objects, so are not very effectively tracked by plane trackers when they are rotated or not facing towards the camera.


Figure 4: Face detection and tracking scheme that uses Viola-Jones facial detection and localization for an initial frame, and a KLT/ORB plane tracking system for each subsequent frame, for each initially found.

#### 3.2.5 Face Blending

Facial blending was achieved via simple alpha blending, in which two images can be blended by applying a weighted sum of pixel intensity for each corresponding pixels. The resultant image is a simple weighted average of the two source images.

For this project, the application specifies a source image in the Android file storage system, from which it extracts a face and bounding box. It then uses this face (and scales it when necessary) to apply alpha blending for all subsequently detected faces.

# 4 Experiments

Because it is not possible to embed video into this report, the results for the following experiments will consist of graph data and example images from the various tracking systems implemented. Some example images will be from test video (as opposed to a live camera feed), because test video can contain a wider and richer variety of lighting conditions, number of faces, and relative camera orientation for testing.

## 4.1 Comparing Tracking Methods

We sought to test accuracy and ability to track faces for the two face tracking approaches described in the technical details section. In Figure 5, we see an examples of facial detection and tracking of the same video for our two facial tracking approaches. Example images were selected to highlight the differences in the two approaches to facial tracking.

In the left panel of Figure 5, we see the results of using Viola-Jones facial localization independently for each frame. The strengths of this method (as shown in the image) are that the system is unable to become "lost" as it is stateless between frames, and the system scales well for many faces because it does not need

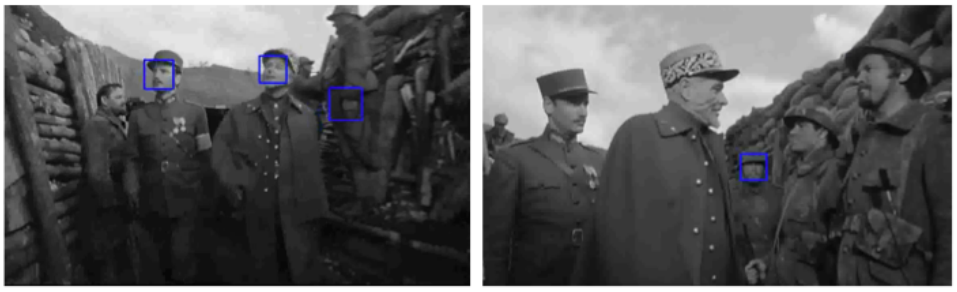


Figure 5: Example of face tracking results from face tracking using independent Viola-Jones detection in each frame (left image) and Viola-Jones initialization followed by KLT and ORB plane tracking (right image).

to initialize data structures for each detected face. However, this system was more likely to detect bounding boxes for regions that did not contain a face. In this example, the rightmost bounding box is actually of a person's knee, and not a face. Since face detection does not incorporate spatial information from frame to frame, this system is more prone to inaccurate detection of faces.

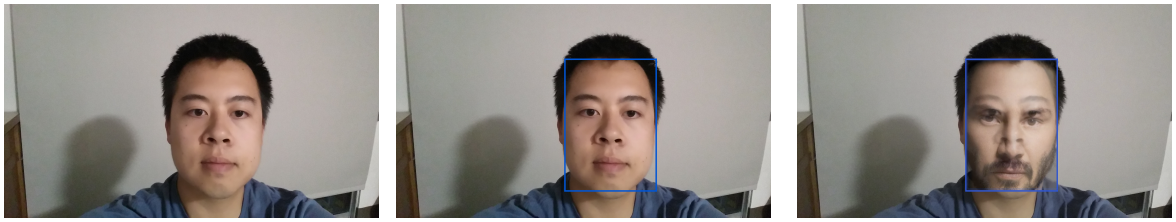
In the right panel of Figure 5, we see the results of using Viola-Jones facial localization for initialization of the tracking system followed by using a KLT and ORB plane tracking system for each face found in the initial frame. This method is less prone to false positives during facial detection, since the spatial constraints applied by a plane tracking system are stricter for determining that a region of interest contains a face. However, this method is flawed because it requires that faces be planar, which they clearly are not. In the example image, we see that the system loses tracking ability for moderately rotated faces, during which face planarity obviously no longer holds. Furthermore, the relatively small bounding box used for KLT and ORB tracking only creates a small number of features to be used for tracking. In previous projects, we found that plane tracking was very effective with more features, so the small bounding boxes will weaken the tracking ability of this system.

| Video | Average bounding box area (pixels$^2$) | Average number of features per face | Percent of frames successfully tracked | Video Description |
|---|---|---|---|---|
| paths of glory | 8190 | 9.5 | 51.4 | A 30-second clip of two men, at a distance, walking while facing the camera, with other people in the background |
| user test 1 | 61918 | 70.3 | 89.1 | A 30-second clip taken with the mobile device of a user face moving up, down, diagonally, away, and towards the camera |
| user test 2 | 16104 | 16.9 | 70.3 | A scaled ($\frac{1}{2}$ in each dimension) version of "user test 1" |

Table 1: Tracking results and face detection properties across different test video feed inputs.

Table 1 shows facial tracking performance across various test input videos for the approach that uses Viola-Jones in the initial frame followed by a KLT and ORB plane tracker. The general trend is that with a larger bounding box for a face (this usually occurs with videos of higher resolution, or if the face is closer to the camera), the system is able to compute more features. With more features to use, KLT and ORB tracking is generally more effective. We can observe this effect by noticing that the percent of frames successfully tracked - as calculated by the fraction of all frames successfully tracked by the plane tracker - increases with a larger facial bounding box and therefore more features per face.

### 4.2 Facial Blending



(a) Original video frame (b) Tracked video frame (c) Blended video frame

Figure 6: Example of video frame across various stages of the application pipeline


Figure 7: Result from multiface tracking on mobile device. We observe that not all faces were tracked in the initial frame, so they are not tracked in subsequent frames by the KLT and ORB plane tracking system.

### 4.3 Measuring Mobile Performance

For this application, the limiting resource on the mobile platform is computational power, because the feature matching involved with the plane tracker is computationally very intensive, especially when a face is lost by the tracker. Furthermore, computation of the cascade of classifiers in Viola-Jones facial detection and localization is not memory limited, especially on the NVIDIA Shield platform, which has a relatively large amount of memory.

Therefore, performance metrics will predominantly focus on computational demands, which can be measured by the output number of frames per second.

In Figure 6, we observe the performance results on a mobile platform for the two facial tracking systems combined with facial blending.

For the system that uses Viola-Jones facial tracking independently for each frame we make the following observations:

1. Performance is relatively steady independent of the accuracy of detected faces.
2. Performance increases significantly when less faces are present in the frame. This is likely due to the fewer computations required during the cascade of classifiers.

For the system that uses Viola-Jones facial tracking for initialization, followed by a KLT and ORB plane tracker, we observe that:

1. Performance heavily depends on the ability of the plane tracker to locate faces. When the system loses track of them, performance drastically reduces.
2. Performance depends on the total number of faces found in the frame at a given moment.

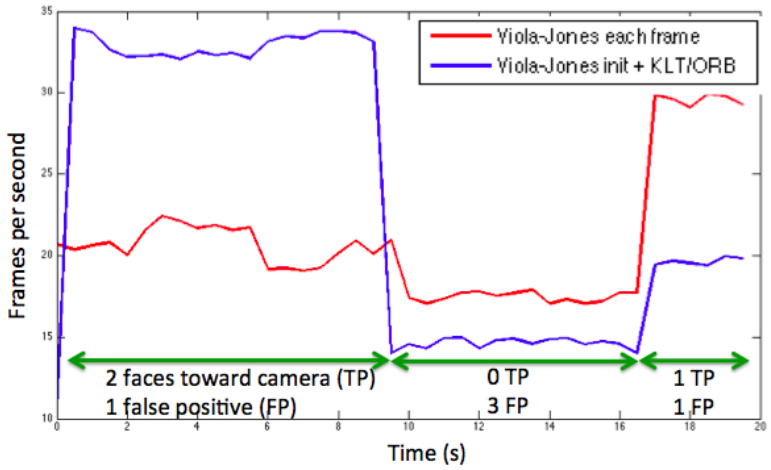



Figure 8: Frame processing performance of two facial tracking approaches across a 20-second video feed with varying number of tracked faces.

# 5 Conclusions

Facial detection, localization, tracking, and blending are feasible tasks on today's mobile device platforms. While performance in this project is not exactly real-time performance yet ( 30 frames per second), we were able to achieve on-par with real-time performance (18-30 frames per second) with a relatively simple design and a few optimizations.

Facial tracking accuracy for our system can still be improved. While most of the frames have faces successfully tracked, frames that contain faces at an angle are often not tracked successfully. To improve this system, we could train the classifier with Haar features on faces that are turned at an angle, which would allow for a more general face detector.

Overall, using a Viola-Jones face detector to identify faces in an *initial frame*, followed by face tracking in subsequent frames with a KLT and ORB plane tracker, is more effective than detecting faces independently in each frame.

Finally, while facial blending is a fun application for users, simple alpha blending (which is easily computed for a mobile device) is not convincing enough as a true "blend" of faces. It may be a while before mobile computing hardware is powerful enough so that a more advanced approach, such as Poisson blending using image pyramids, is feasible.

# References

[1] Amscreen. *OptimEyes FAQs*. Amscreen Group Limited, 2014.

[2] Viola, Paul. Jones, Michael. *Rapid Object Detection using a Boosted Cascade of Simple Features*. Conference on Computer Vision and Pattern Recognition. 2001. Cambridge, MA.