Comp Photography Final Project

Douglas Mead

Summer 2016

dmead3@gatech.edu

Emoji Face

Emoji Face is an iOS app that uses the front facing camera to detect emotions, map that emotion to an emoji, then warp / transform the emoji image to the user’s face.. in real time.

The Goal of Your Project

This project had a few goals. One, I wanted to apply what I learned in this class to an iOS app. Two, I wanted to attempt to use OpenCV in Swift (this was a huge obstacle). Three, I wanted to explore different SDKs and learn about facial recognition. Lastly, the goal was to do this in real time.

Scope Changes

- Originally, I wanted to make this as one of the new iMessage apps, however I had issues with iOS 10 Beta, so did not.
- It was not listed in the proposal, but I thought I would be using OpenGL. The learning curve proved to be too steep for this timeframe, so I ended up using OpenCV.
- In case the scope of this project was too small, I had proposed an idea to extend this past just facial emojis. This became unnecessary.

Input

.. and others

Output

In Progress

Final

https://drive.google.com/open?id=0BxmMkNi5YufZR3B4WGwtZFVhTFU

Your Pipeline

Determine "Base Points" on Emoji Images

Input

Camera Feed
Emoji Image Set
Emotions

Detect Faces and Emotion

Calculate Homography

Perspective Warp Emoji to Camera Feed image

Output

- The input comes from:
  - A set of emoji images from emojione.com
  - Camera stream processed using Affectiva
- The Affectiva framework outputs image frames and an array of faces and emotion data
- Determining the "Base Points" was done using Python code to determine averages
- Using these "Base Points" and the points given from Affectiva, OpenCV functions calculated homography and warped the image

Demonstration: Show complete Input/Output results

This video demonstrates the app in use:

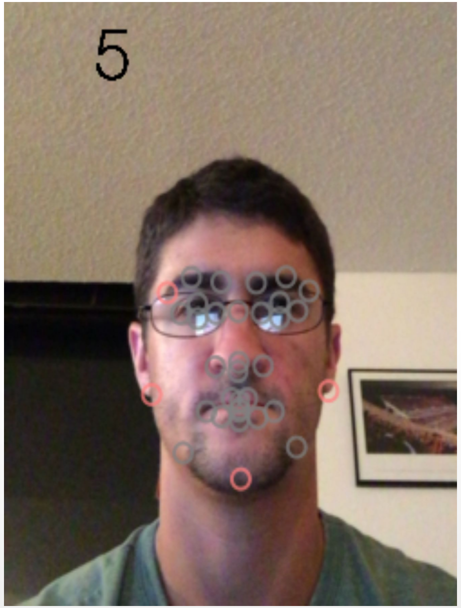
https://drive.google.com/open?id=0BxmMkNi5YufZR3B4WGwtZFVhTFU

Emotion Detection

- After much research, I chose to use a trial of an SDK by Affectiva (http://www.affectiva.com/). They make software for use advertising to determine emotion and reactions.
- Other options are linked here (they all had pros and cons, however Affectiva was free, had a native SDK, and demo project)s:
  - Google (https://cloud.google.com/vision/docs/)
  - Emovu (http://emovu.com/e/)
  - Microsoft Emotion API (https://www.microsoft.com/cognitive-services/en-us/emotion-api)
  - Face++ (http://www.faceplusplus.com/dev-tools-sdks/)
  - FaceWrapper (https://github.com/sergiomtzlosa/faceWrapper-iphone)

Affectiva Points

- The documentation did not specify which points were which, so I wrote a program to manually alternate through the points and write down their corresponding index. Fortunately the indices stayed consistent.



Average Base Points

- In order for homography to work, I needed what I called base points. These are the points for an unwarped emoji image (facing forward). I initially picked five: two sides of the jaw (ears?), bottom of the chin, and outside of the eyebrows. These formed a roughly home plate shape that I thought would be good for homography. To get the best results, I looked straight at the camera and printed the results multiple times.
- Next, I put these in a .txt file and wrote a small python program (https://github.com/dmead28/EmojiFace/blob/master/findAve.py) to determine the average values, normalized from 0.0-1.0 so that it would work with different sized images.

Homography and Warp Perspective

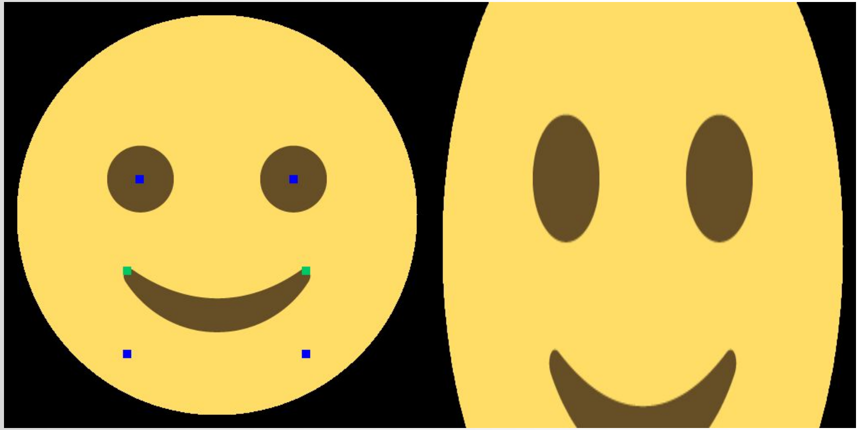
- Since this class gave me comfort using OpenCV with Python, I used it for prototyping and then translated my Python code to Objective-C++
- This is the non-trivial part of the code:

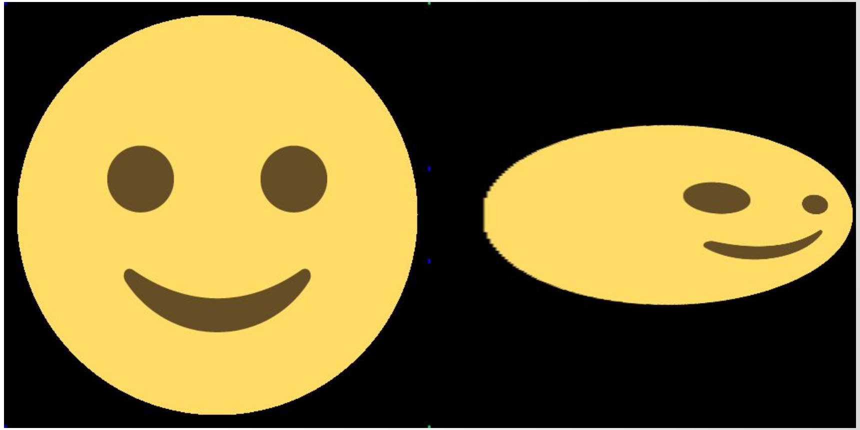
```
# Calculate Homography
h, status = cv2.findHomography(originalPoints, newPoints)
# Warp source image to destination based on homography
newImage = cv2.warpPerspective(image, h, (image.shape[1],image.shape[0]))
```

(https://github.com/dmead28/EmojiFace/blob/master/homographyCV.py)

Homography and Warp Perspective

- Using the Python code mentioned on the previous page, I was able to experiment and get a better grasp on homography.





OpenCV in iOS (Swift)

- Using OpenCV in the Swift project was probably the most difficult part of the project. OpenCV is written in C/C++ which both work well with Objective-C. Additionally, Swift works well with Objective-C. Going from C/C++ to Swift, however, introduces some bugs. To get around this, I had to wrap the functionality of OpenCV in an Objective-C++ .mm file. To get around some issues, this was just a class with static methods.

(https://github.com/dmead28/EmojiFace/blob/master/EmojiFace/OpenCVWrapper.mm)

OpenCV in iOS (Objective-C++)

```
+ (cv::Mat)cvMatFromUIImage:(UIImage *)image {

    CGColorSpaceRef colorSpace = CGImageGetColorSpace(image.CGImage);
    CGFloat cols = image.size.width;    CGFloat rows = image.size.height;

    cv::Mat cvMat(rows, cols, CV_8UC4); // 8 bits per component, 4 channels (color channels + alpha)

    CGContextRef contextRef = CGBitmapContextCreate(cvMat.data,                 // Pointer to  data
                                                    cols,                       // Width of bitmap
                                                    rows,                       // Height of bitmap
                                                    8,                          // Bits per component
                                                    cvMat.step[0],              // Bytes per row
                                                    colorSpace,                 // Colorspace
                                                    kCGImageAlphaPremultipliedLast | kCGBitmapByteOrderDefault); // Bitmap info flags

    CGContextDrawImage(contextRef, CGRectMake(0, 0, cols, rows), image.CGImage);
    CGContextRelease(contextRef);
    return cvMat;
}
```

OpenCV in iOS (Objective-C++)

- The previous code converts a UIImage (iOS) to cv::Mat (C++)
- Boiler plate code was taken from the OpenCV documentation at: http://docs.opencv.org/2.4/doc/tutorials/ios/image_manipulation/image_manipulation.html
- Using the constant: kCGImageAlphaPremultipliedLast helped solve some issues I was having with alpha values.

OpenCV in iOS (Objective-C++)

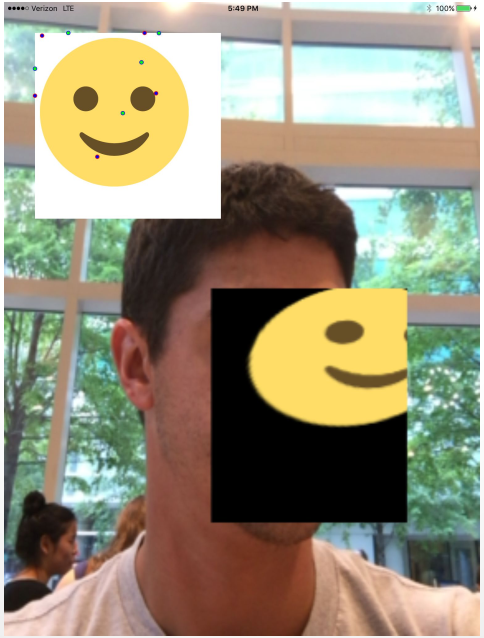
```
+(UIImage *)warpSmiley:(UIImage *)originalImage fromPoints:(NSArray *)basePoints toPoints:(NSArray *)points usingSize:(CGSize)newSize {
    // Some code cut for space
    cv::Mat image =  [self cvMatFromUIImage: originalImage];
    std::vector<cv::Point2f> srcPoints, dstPoints;

    for (int i = 0; i < basePoints.count; i++) {
        CGPoint point = [[basePoints objectAtIndex: i] CGPointValue];
        srcPoints.push_back(cv::Point2f(point.x,point.y));
    }
    // … same for dstPoints

    cv::Mat homography = findHomography(srcPoints, dstPoints);
    warpPerspective(image, newImage, homography, cvSize);

    return [self UIImageFromCVMat: newImage];;
}
```

OpenCV in iOS (Objective-C++)

- The previous code was ported from the Python file into the hybrid Objective-C++ language.
- The code is pretty straightforward, it computes the homography matrix and then uses it in OpenCV’s warpPerspective function.
- These map pixel locations to their corresponding warped locations. This is an operation in domain, rather than range (as in filters)
- I chose not to use an affine transform because the points given from the Affectiva framework would not be restricted to parallel lines.

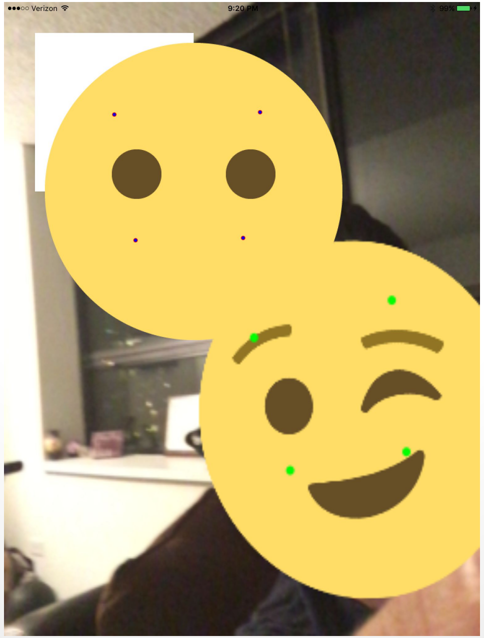
Initial Warping in Project

- I created a UIView to visualize the points and diagnose bugs such as reverse coordinates (x,y -> row,column), image size issues, etc.



Correcting Points

- After experimenting, I determined that the points I used were being inaccurately tracked. I ended up switching to more reliable points near the eyes and cheeks.



Details: What worked?

- Overall the project was a success! I definitely spent more time than expected on getting interoperability between Swift and OpenCV. This was something I had been meaning to learn (don't have much C++ experience), so it was a win.
- The emotion detection (although sometimes spotty) worked very well and the framework even mapped to specific emojis. All I had to do was download images and map file names to integer constants in the C++ enum (https://github.com/dmead28/EmojiFace/blob/master/EmojiFace/ImageAssets.swift) and (emojione.com)
- Warping to a person's face worked well. The emoji turns with the orientation of the person's face.

Details: What did not work? Why?

- Delaunay Triangulation and OpenGL: I wanted to create a 3d model drawing from http://stackoverflow.com/questions/20863904/warp-bend-effect-on-a-uiview and https://www.learnopencv.com/delaunay-triangulation-and-voronoi-diagram-using-opencv-c-python/ and http://davis.wpi.edu/~matt/courses/morph/2d.htm
  - I was not able to learn OpenGL in time. To replacate this, I had the idea of using homography to warp triangles manually.
  - OpenCV requires four points for affine and perspective warps. Using three (four with two very close) points lead to some issues and this became complex, so I abondoned it.
  - I've ommitted details, although a significant amount of research was put into it.
- As previously described, using the jaw and chin points did not work. This worked well when turning left and right, but not up and down. When looking down, the points spread to my ears (sometimes out to my shoulders) and gave the perception that the smiley was looking in the opposite direction.

Details: What would you do differently?

I’ve discussed this previously, but I would have really liked to turn this into a 3D model using OpenGL. The emoji would more closely match the person’s facial features and would look more like a mask. Some blending would also look pretty cool.

Resources

- http://emojione.com/
- http://stackoverflow.com/questions/20863904/warp-bend-effect-on-a-uiview
- https://www.learnopencv.com/delaunay-triangulation-and-voronoi-diagram-using-opencv-c-python/
- http://docs.opencv.org/2.4/doc/tutorials/ios/image_manipulation/image_manipulation.html
- http://stackoverflow.com/questions/29616992/how-do-i-draw-a-circle-in-ios-swift
- https://github.com/Affectiva/ios-sdk-samples
- http://www.learnopencv.com/homography-examples-using-opencv-python-c/
- http://docs.opencv.org/2.4/doc/tutorials/ios/image_manipulation/image_manipulation.html
- https://www.learnopencv.com/warp-one-triangle-to-another-using-opencv-c-python/
- http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=findhomography#findhomography
- http://www.learnopencv.com/homography-examples-using-opencv-python-c/
- https://www.learnopencv.com/face-morph-using-opencv-cpp-python/
- https://alliance.seas.upenn.edu/~cis581/wiki/Projects/Fall2013/Project2/Proj2-2013-Fall.pdf
- http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture/
- http://davis.wpi.edu/~matt/courses/morph/2d.htm#3
- http://people.scs.carleton.ca/~c_shu/Courses/comp4900d/notes/homography.pdf

Your Code

All code is hosted at:

https://github.com/dmead28/EmojiFace/