Brennon York
18 July, 2017

Report

1. Display Feature Points

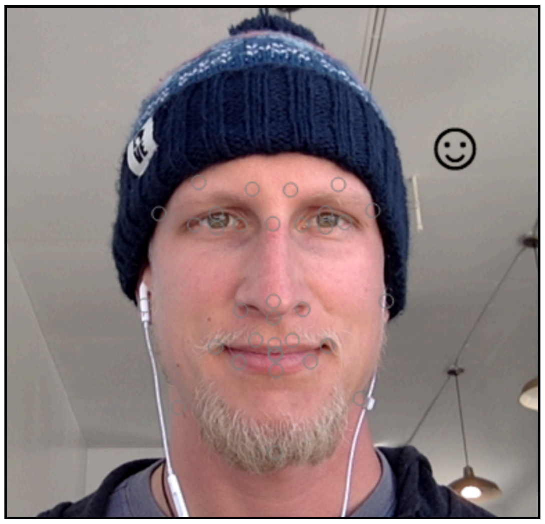


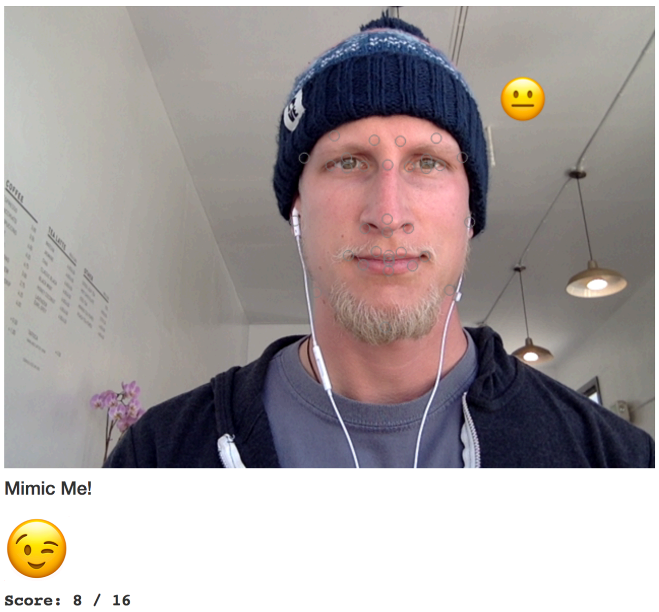
Figure 1

In this section I needed to display the unique points on the face that the Affectiva API uses to determine what is a face. Here we're given a face object with `faceFeature`s that are an array of X and Y coordinates for each feature. What we needed to do was leverage those X and Y coordinates along with the `arc` call to draw radii around those points. I chose to make the points gray in color and small to easily be seen by the application. You can see this below on Figure 1.

2. Show Dominant Emoji

Here we needed to anchor the dominantly displayed emoji that is given to us from the Affectiva API onto our displayed face. The emoji is given from the `face` object at `face.emojis.dominantEmoji`. From there it was a bit of trial and error in leveraging the both the `fillText` and `strokeText` APIs to display the emoji. At first I was using Safari with the `strokeText` API and no emoji was displaying. I switched browsers to Chrome which didn't change anything. Then I tried the `fillText` command which finally displayed the emoji. From there I anchored the emoji to the first facial feature in the `faceFeature`s list and offset the emoji by adding static values to both the X and Y points. This works really well and the emoji easily tracks the face. If I were to spend more time however I would add bounding boxes to ensure that, if the face is on the edge of the screen, the emoji doesn't go outside the video box which it can do currently. You can see the emoji output on Figure 1. The default smile face displays oddly unlike the other emojis, something I would also look into if I had more time.

3. Implement Mimic Me!


Mimic Me!

😉

Score: 8 / 16

Figure 2

The last portion of the project was to implement the Mimic Me! game which entails having the user mimic their facials expressions to match that of the emoji given. If they do, their score goes up. I chose to implement this as a timer-based game having an individual try to match the emoji within 8 seconds and, if not, the emoji will switch. One thing I added was to make sure that, when an emoji switches, it never switches to the one that it already was ensuring that a user couldn't get lucky and hold the same facial features and continuously land on the same emoji. This was done by creating a simple clone of the default `emojis` array, removing the emoji that is already present, and then choosing a random emoji from the rest of the possibilities within the array. The game also keeps track of the user's score and displays in at the bottom of the screen along with the current emoji to match. If the player hits the reset button the game will reset back to zeroes on the score and timers. If I had more time I would add a timer to the screen in the middle to count down the beginning of the game as well as a small timer in the corner to display the amount of time remaining to match the current emoji. You can see the game being played in Figure 2.