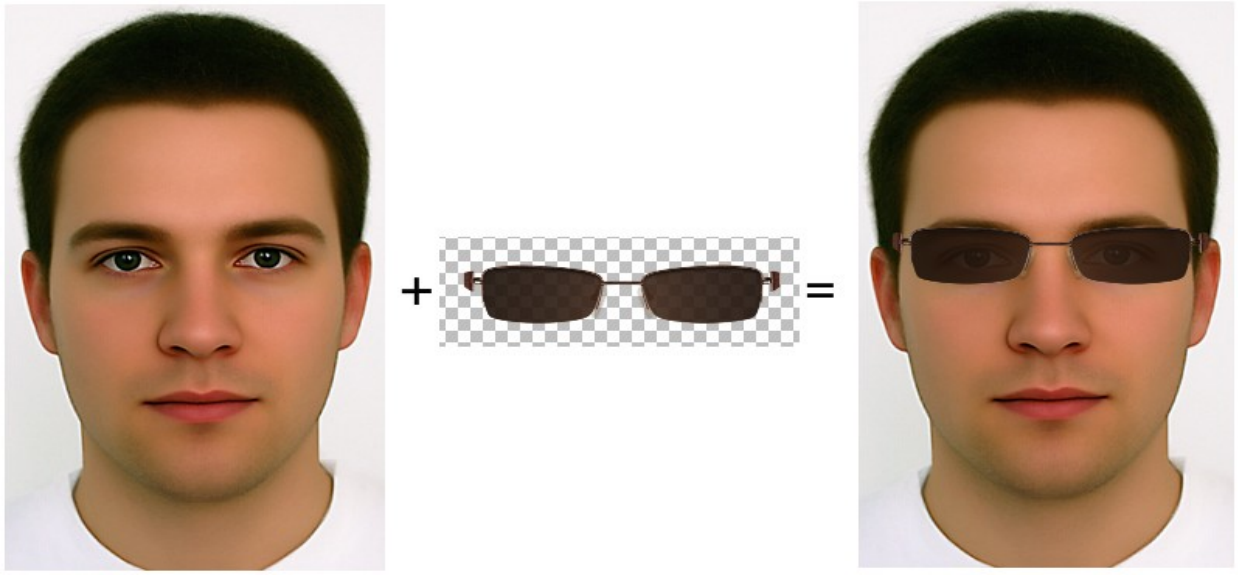
Overview:

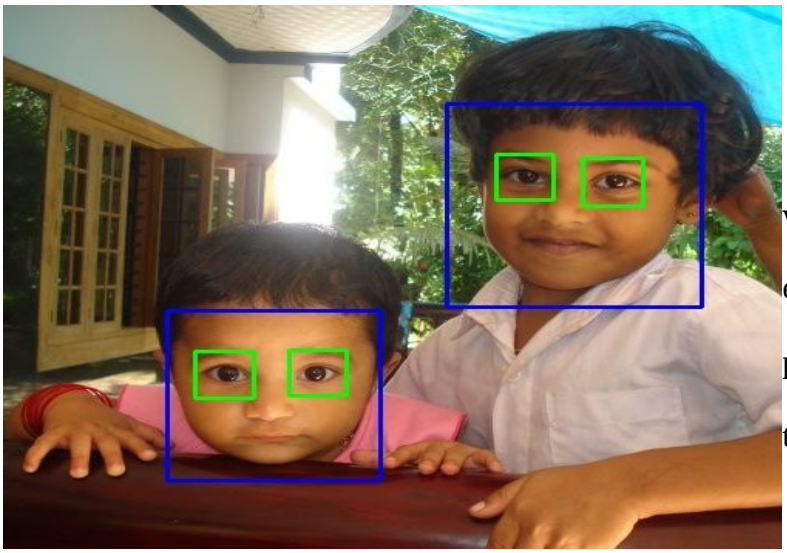
Our project aimed at 3 objectives: to be able to trace a face with relatively good accuracy, to overlay a simple set of images over that face, and to create an phone application that lets you interact with the video. However, after really getting into it we had to narrow down the scope of our project, cutting down our third objective down to just the bare bones of a full desktop application. The result of our efforts is that being able to select a video file interactively and selecting the assets; which are restricted to the eyes and mustache area of the face. Also, there is a very small semblance of object tracking as the user is able to select a region of interest by pausing the video and clicking on 4 points that are shown in a second window, however, this did not end up as fully flushed out as it should have been and often does not track accurately the selected region.

Background:

Ever see the watermark on your TV screen of the station you are currently watching? Or maybe a spoof video with another image or video super imposed in it that doesn't quit belong? These are example of video overlaying. There are many applications for video overlaying, but the target of our project was to be able to overlay a face or faces in a video, pick some fun glasses and a mustache, and place them over the face and have it move along with the face as best a possible.

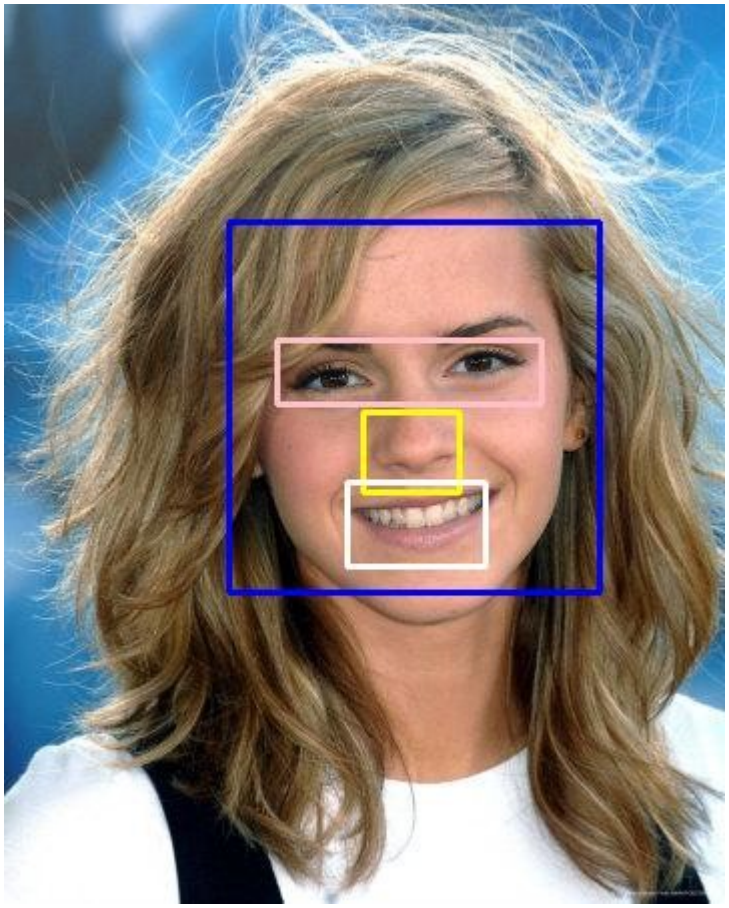


The above images are an example of what we tried to accomplish. However, unlike the simple representation above, it is not always easy to track faces, and even less so when tracking individual features of the face such as the ears, eyes, nose or mouth because they come in many different proportions and are much smaller than the overall size of the head. Fast movement and lighting can make a huge difference while tracking them. Fortunately, there are some ways to track faces, using a set of XML data comprised from a set of images called a Haar-cascade classifiers. Theses data files are the result of the identification of many features a set of images, which can in turn be used to identify similar features in any other given video or images with some degree of accuracy. In our project, we used these haar-cascades to find and track a few of the features: like the face, eyes, and mouth. Here are a few images, that show what we were tracking:



This image comes form the openCV website, and is an example of what an example program they provide tracks. This photo shows the use of haar-cascades that track individual eyes and the face.

This picture of Emma Watson shows another way of tracking the eyes as a whole. In addition to the eyes and face it also shows the ability to tracking the nose and mouth as well



Being able to track these features can be of vital importance in may fields such as security for identification from a video camera and in more civilian friendly fields such as virtual reality.

Methods:

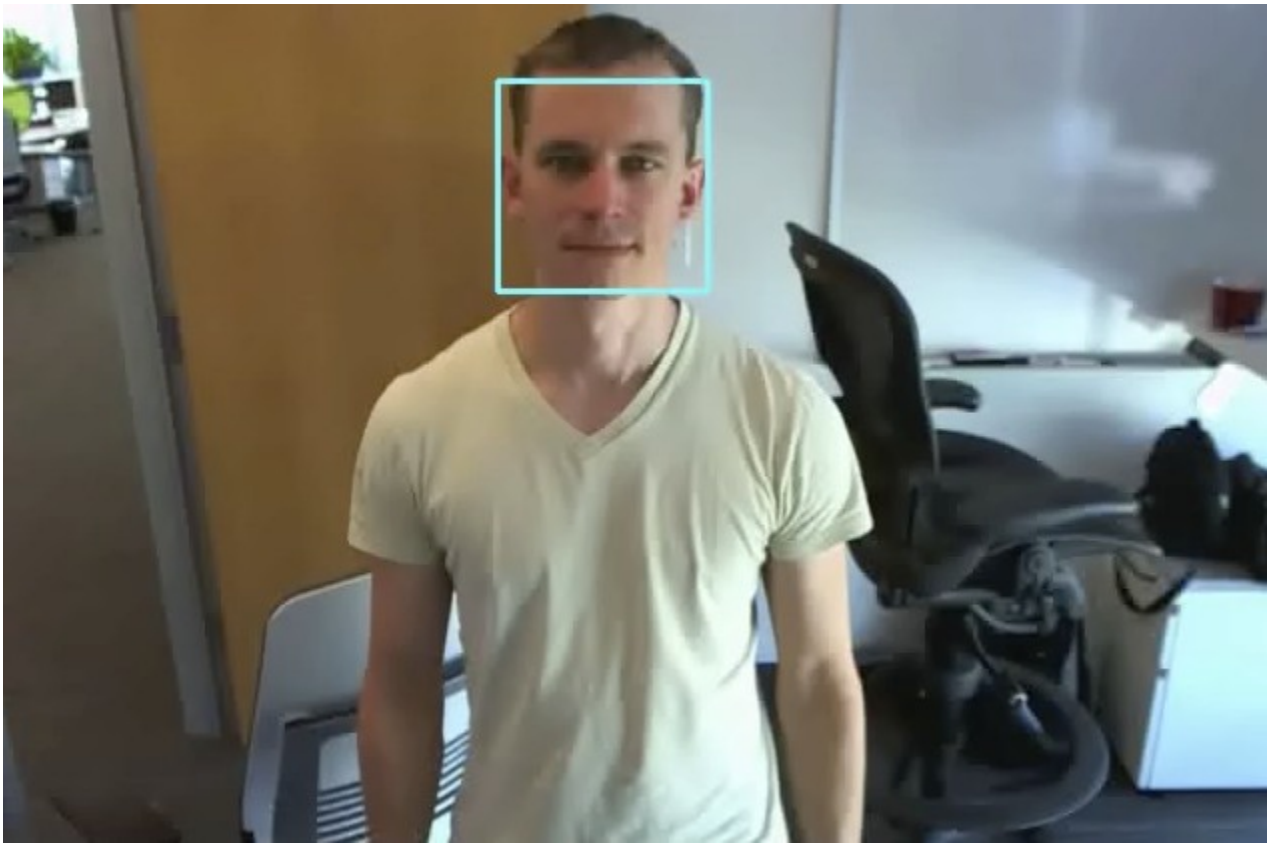
Our methods are rather simple, as most of the difficult features came from trying to create and application for it to work with rather than the act of tracking the videos through code itself. But our implementation did involve having to use a few algorithms in order to maintain accuracy between the frames of the video and stabilization of the overlaid images. For example, when tracking the smaller features on the face, in order to prevent from getting false positives, we had to narrow the scope of where to look. We made sure to only look inside the detected face for these features, less it detect a feature nowhere near the actual face and cause problems in being able to track it. Also, we used some Markov addition in our implementation to attempt to stabilize the image. In our attempted implementation of object tracking we use a Camshift algorithm to attempt to detect the difference between frames and thus keep track of an object as the video progressed, but it is broken, most likely due to some of the hues present in the video throwing off the color detection.

Results:

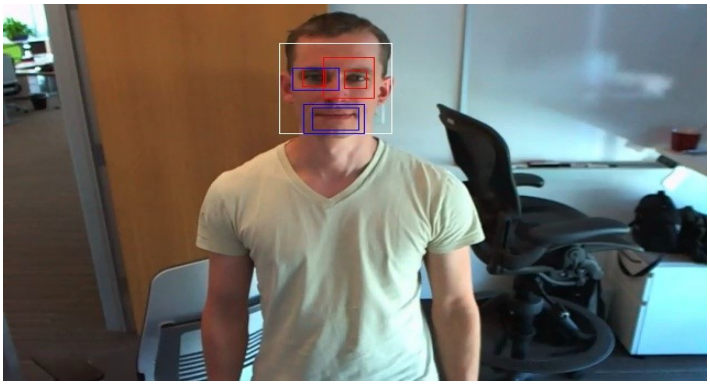
The results of our first objective were relatively simple at first, as in another project we were already able to track a single face with at least 80% accuracy. According to the unit test run on it the results where

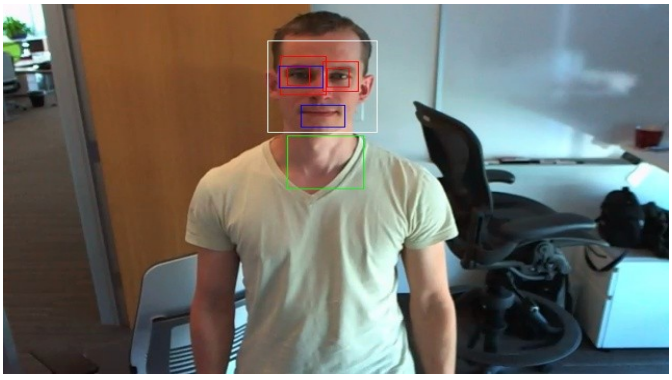
Stats from unit test of our face tracker vs the grounded face tracker :

Total of 3521 pixels of bounds difference across 327 frames from the following video:

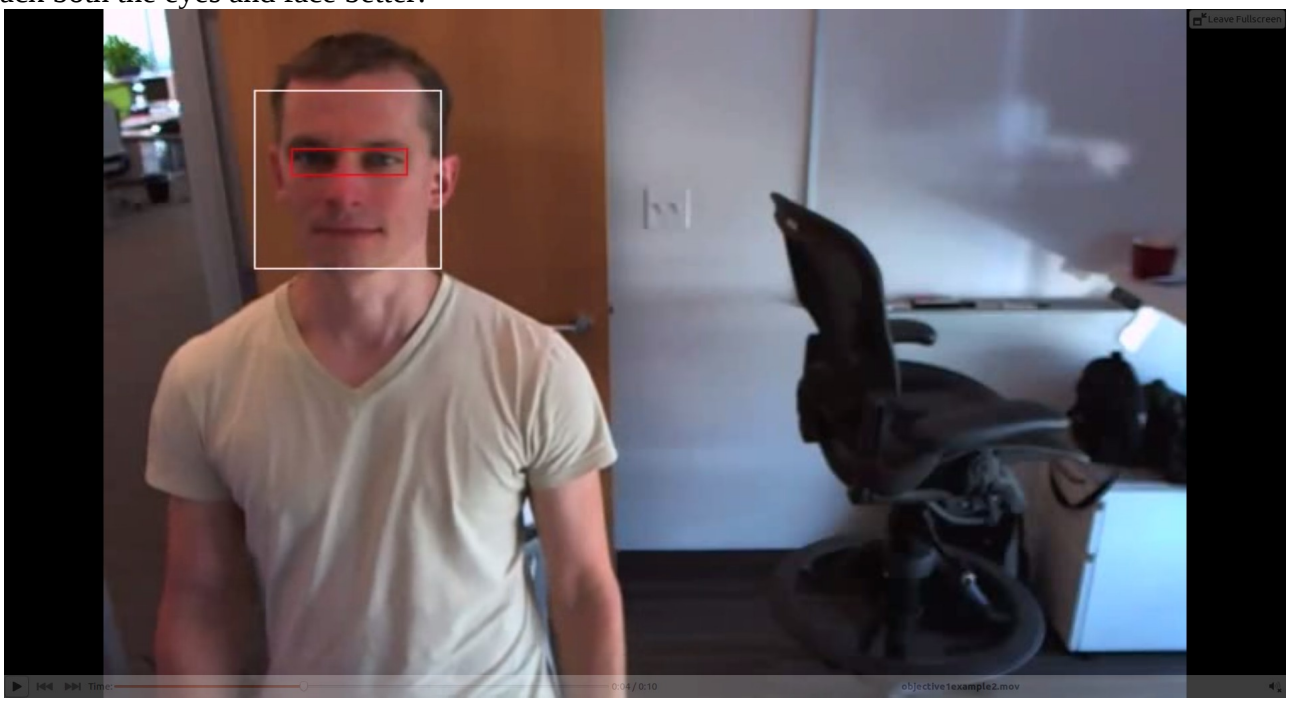


However, as we added more features to track things got a little more complicated and we began to lose accuracy. Here are some of the images that we took when we tried to track more features on the face:

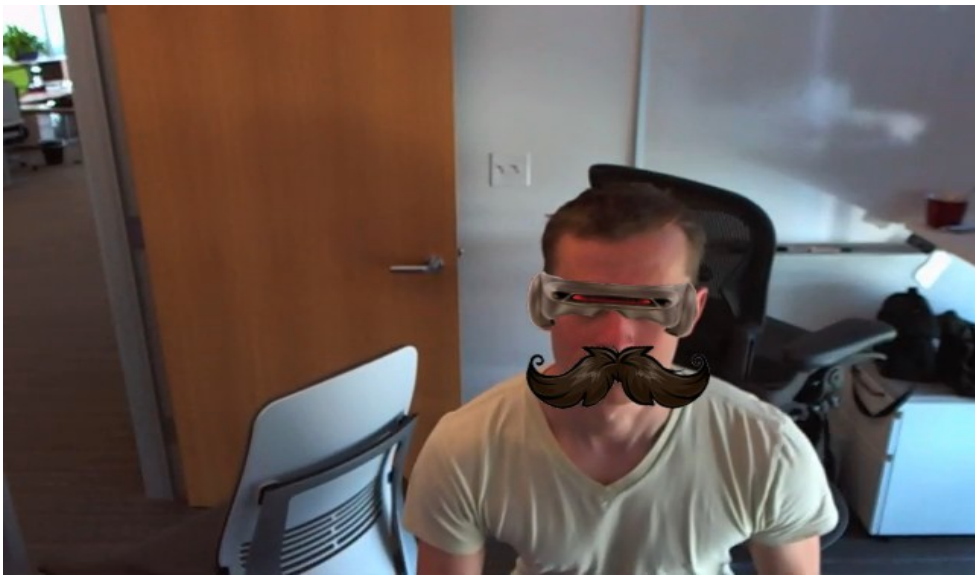




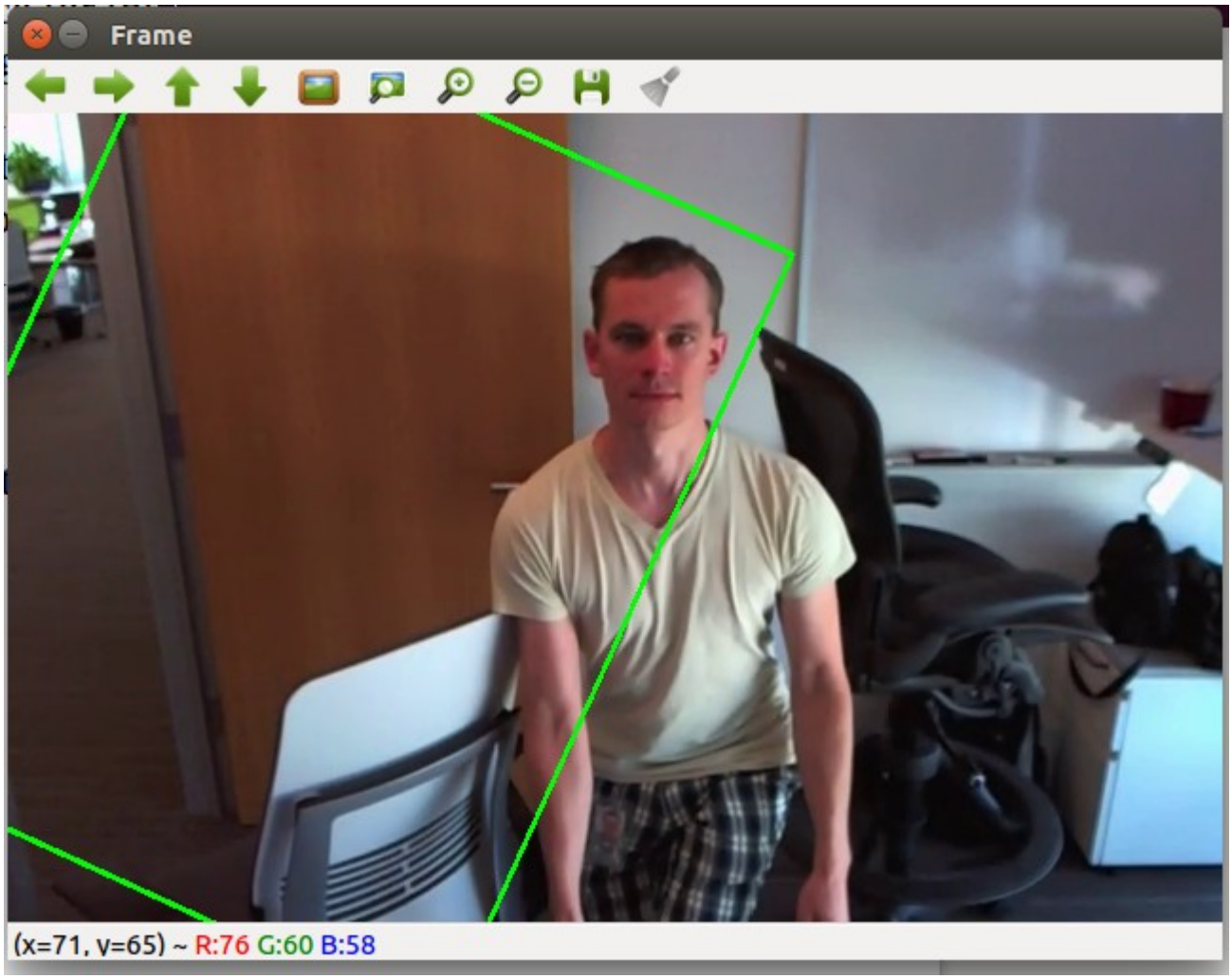
These images were the result of not narrowing down the search area for the features before detection and thus would return false positives. However after some correction we would be able to track both the eyes and face better:



our result from objective two are better seen in video but here are some images of what it looks like:



As for our final objective, we are able to interactively select a ROI by pressing 'I' then clicking 4 points in the “frame” window, and then let the thing continue, however, it is very inaccurate. Here is a picture of what happened to the box later in the video after it started around the face:


Frame
(x=71, y=65) ~ R:76 G:60 B:58

Sources:

Images:

https://realpython.com/blog/python/face-recognition-with-python/

http://docs.opencv.org/trunk/doc/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html

http://www.imadm.ca/programming/2013/01/18/facial-overlay-with-pyopencv.html

Source Code:

https://realpython.com/blog/python/face-recognition-with-python/

http://www.imadm.ca/programming/2013/01/18/facial-overlay-with-pyopencv.html

http://tkinter.unpythonic.net/wiki/tkFileDialog

http://zetcode.com/gui/tkinter/introduction/

http://stackoverflow.com/questions/8384737/python-extract-file-name-from-path-no-matter-what-the-os-path-format

http://www.computervisiononline.com/blog/tutorial-using-camshift-track-objects-video