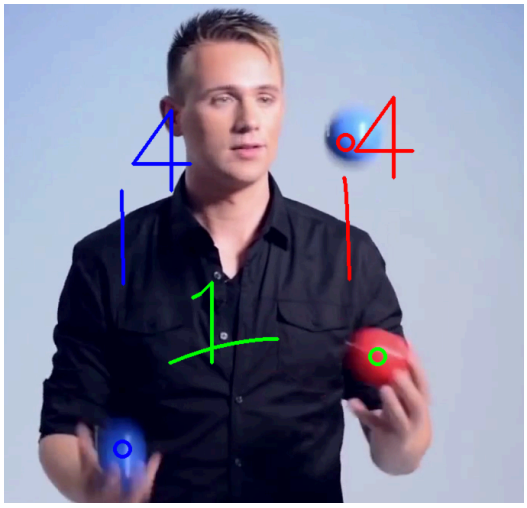
**CMPS 161 Final Project**
By Cole Faust

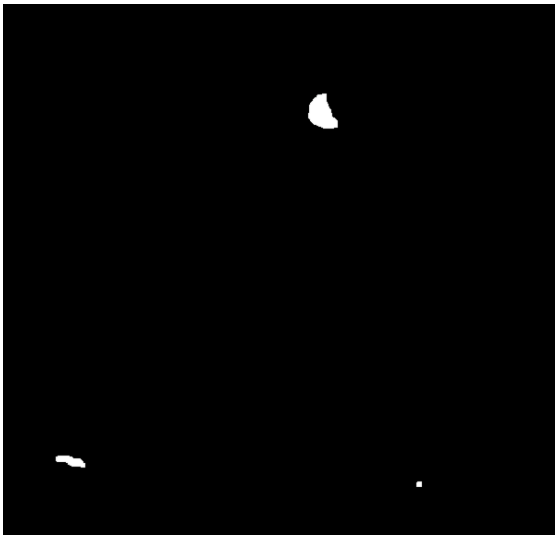
**Introduction**
For my final project in our Intro to Data Visualization class, I made a program that would analyze a video of someone juggling and attempt to figure out what throws they're making. The data visualization part comes from drawing labeled arcs over the video. The end result looks something like this:



Arcs are drawn over the path that each ball makes, and those arcs are labeled with the siteswap notation of the throw. The arcs are quadratic regressions from the points given in the video, to create a smooth line and remove irregularities.

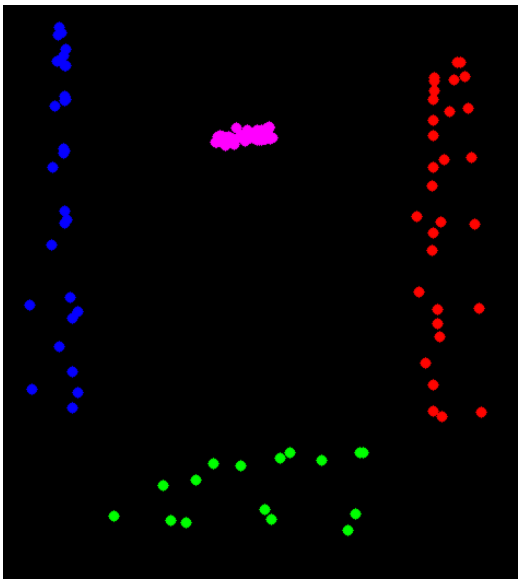
For those not familiar with siteswap, it is a notation used to describe juggling patterns. Each throw of the ball has a number associated with it, where odd numbers are increasingly higher throws from one hand to the other, and even throws are increasingly higher throws from one hand to itself. Two small exceptions to this rule is a 2 in siteswap corresponds to holding the ball without throwing it, and a zero corresponds to not having a ball in play for that beat. A combination of these numbers can make a complete juggling pattern. In the demonstration video I showed for this class, there were two different patterns: one made of 4's and 1's, and one made of only 3's.

**Implementation**
I wrote this a C++ program with opencv to do the tracking, and GNU scientific library to do the regression. Since this was my first foray into computer vision, I ended up making many assumptions about the video in order to make it easier to track the balls. Most notably, I assumed everything with a saturation of less than 150 or value of less than 100 (in HSV color format with values ranging 0-255) was in the background and could be removed. That left pretty much just the balls for the video I chose, as it had a very light background and a black shirt. Remaining small fragments could be removed by eroding the image. Here's an example of what the resulting image looked like:



This process left the balls not looking very ball-like however, so I picked them out by finding any non-zero pixel and then using flood fill to remove it from the image while I look for other balls.

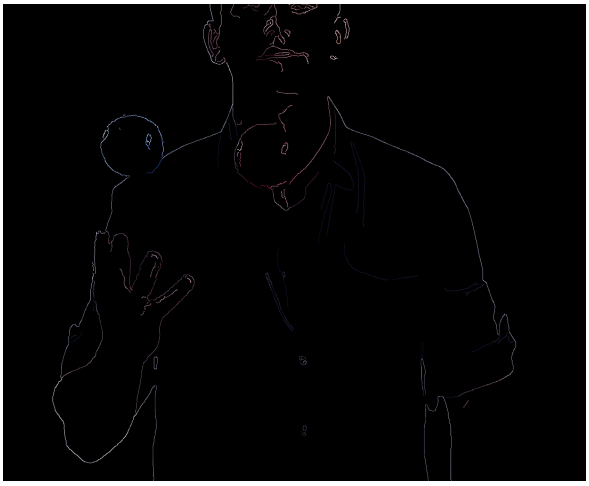
After identifying the locations of the balls on this frame, I try to figure out which ball matches up with another from a prior frame. What I went with was just a simple “pick the closest” choice. In the videos in my demonstration, balls rarely cross over each other, so it’s reasonably accurate. I have 3 cursors that I move along with their closest ball, and where the cursors are I record points as part of the arc of that ball. While recording those points, if any of them fall outside of a certain range, I consider the arc ended, and subsequent points will start a new arc. The range of valid points I define to be halfway between the juggler’s face and the bottom of the video, or roughly underneath the face. (to catch the 1’s) It’s not a very good system, but again, I was just trying to get it to work for this one video. An image of the plotted points looks like this:



The pink points are the juggler’s face, and the other colored points are where the balls are.

**Other Attempted Strategies**

Although the final product is very dependent on the input data, I had tried some more general strategies earlier. One of those strategies was to detect where the balls are by using Huffman Circles. Huffman Circles are a way to detect circles in an image, that OpenCV has native support for. They work off of finding the contours in an image and then figuring out how much of the contour matches a circular shape. If that value passes a certain threshold, it’s considered a circle. The problem with this was that contours are very general, and they could be applied all across the image. Here’s an example of contour line output.



As you can see, it can be confusing even to a human to tell what’s going on. You can see the blue ball pretty well in this image, but the red ball is only half there, and it’s mixed in with the juggler’s neck. This produced very inconsistent results when applied to video. It often thought that there were several balls in the juggler’s face and shirt, didn’t pick up the real balls sometimes, and even thought the circular logo on the video was a ball.

Another strategy I tried to use was optical flow in order to figure out which ball was which. Optical flow matches objects with prior objects not just with position, but also velocity and acceleration. This may have worked well, and I started implementing the basics of it, but I discovered that just doing a simple proximity check worked out well enough for my purposes, especially because in the juggling patterns I analyzed the balls don’t cross or overlay over each other. For more

complicated patterns or patters with more balls this may be necessary.

Finally, to remove the remaining scraps of the image that wasn't part of a ball, I had tried using a Gaussian blur before eroding, as I had read that it could help. It actually ended up having a negative effect, and caused less of the image to be removed by the later erosion. In hindsight, doing the Gaussian blur after the erosion wouldn't have hurt but it wasn't necessary because I was just detecting any non-black pixels at that point.

**Results**

The final results of the project end up working pretty well for the cherry-picked input data. With more time and research into the proper way to do computer vision, this could end up becoming a useful tool. Some potential applications for this would be teaching new jugglers by clearly analyzing professionals, or showing amateurs what they did wrong. Future goals for the project could be identifying mistakes, creating a complete sight swap notation from the individual throws, identifying the colloquial name for the pattern, or overlaying a sample animation of the pattern. These are all things to think about when moving forward, as this project is just the basics of what it could be.