The Olin Orchestra-less Conductor

By Justin Kunimune and Eleanor Funkhouser

Overview
Our project uses computer vision to allow the user to act as the conductor for a song. The song has four different components, and by moving two batons around in front of their webcam, the user can control the volume on a running basis for each of the four different parts. While the program runs, they can see the webcam feed (divided into four columns for each of the parts) so they know which baton is controlling which part of the music, and move them up and down accordingly to manipulate volume.

Results
Our final product is a program that displays footage from the webcam, divided into four columns, and tracks two brightly-colored batons. The height of each baton maps to a volume such that moving the batons up and down in each column will control the volume of that part. The full song is just over two minutes long.

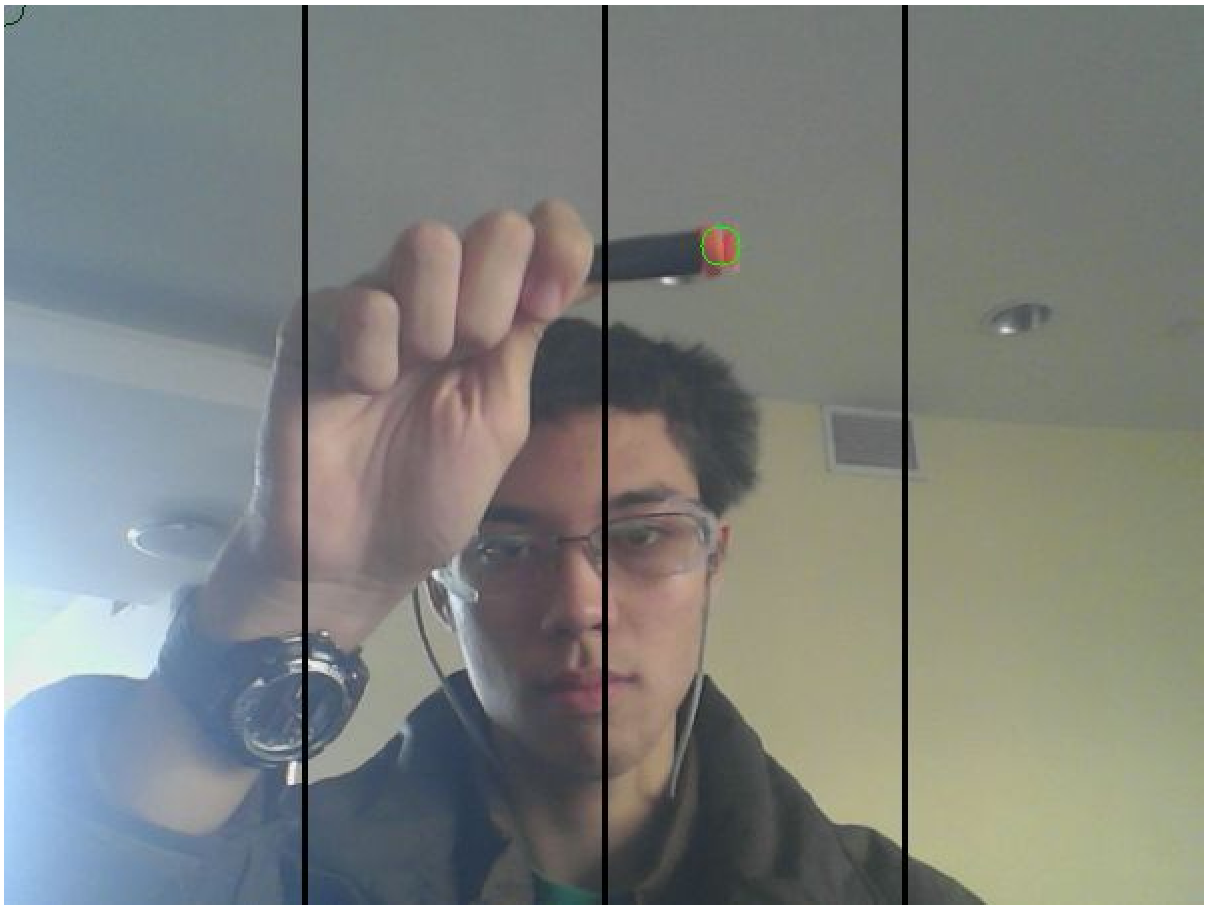

Figure 1. What a user will see as they conduct. The code shows the webcam video stream as well as the four columns and a circle over the brightly-colored head of the baton to validate that it is tracking that baton.

It is possible to use one or two batons, and the code allows for easy addition of even more if the user desires.

Implementation

Our project consists of a single .py file with two classes: Color and Frame (shown below in Figure 2). Color is a simple utility method used to manage color ranges, and Frame contains all of the information that is processed and stored from the webcam (the location of a baton, the volumes of the tracks, etc.). These classes are called from a script that instantiates two Frames and some pygame music, then continuously feeds the Frames information processed from the webcam and uses the Frames to calculate volumes for each of the tracks. The tracks are played simultaneously at their respective volumes using pygame's sound capabilities. Finally, the program displays the current webcam image, highlighting where it thinks each of the batons are and showing which columns are which. All image processing is done with various opencv commands to clean up the image, isolate certain colors, mask out everything but those colors, and track their locations.

One design decision we faced was what kinds of motion/physical commands we would have our code interpret. We initially wanted to have the software take in "natural" conducting motions (the big baton motions, the relative symmetry between two batons, a general continuous hand-waving that conductors typically do to communicate tempo to their orchestra). Perhaps with more time we would have conceived of a reliable and intuitive way to translate these motions to volume levels for each of the parts. However, during the timeframe for this project, it was much more straightforward to just tell the user where each part of the orchestra was and make it very simple and clear to control the volume for each.

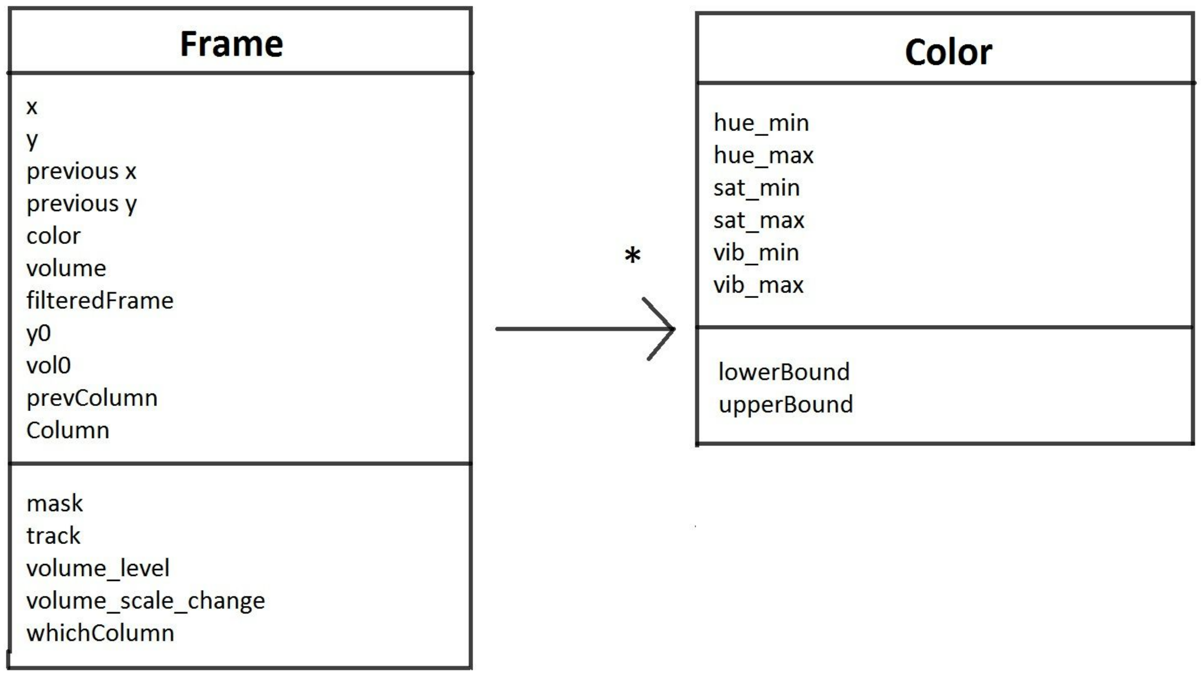

Figure 2. The UML state diagram for the system. Shows the two classes, Frame and Color, where Frame HAS-A Color. Frame performs color-based masking on the video image, extracts/tracks the baton location from that, and contains methods to adjust the volume based on the baton's column and height. The class Color tracks HSV (hue, saturation, and vibrance) attributes to allow creation of an allowed 'range' for a particular color.

Reflection

Process-wise, our project went fairly smoothly. Although we did not accomplish our original goal of controlling the tempo of the music via the movement of our hands, we did manage to work in music with multiple parts, which was not one of our original goals. We started the project with one of us learning about video and opencv, while the other looked into audio and pygame. Once we could track objects, play sounds, and control volume, we worked on putting the code together and optimising it. Whenever one person finished a task, we would look for whatever functionality we wanted to work on next.

We ran into some difficulties towards the second half of the project since only one of us had a functioning webcam, so only one of us could test. We also had to rewrite much of our code when we fused image tracking with sound playback since none of the original code was object-oriented. However, since most of our code was built-in functions from opencv and pygame, it was fairly easy to transition from scripts to objects. One of the biggest time-investments of the project was adjusting tempo. We found a library to do it, but it needed to be installed via a python script in a tar.gz file, and that script required four other libraries that also needed to be installed from a tar.gz, not all of which could be easily found on the internet, so we eventually gave up on the idea in favor of controlling other features of the song that would not take so long to implement.