Aruco Marker Synthesizer

Alexander Tedrow
atedrow@clemson.edu
CPSC 4820

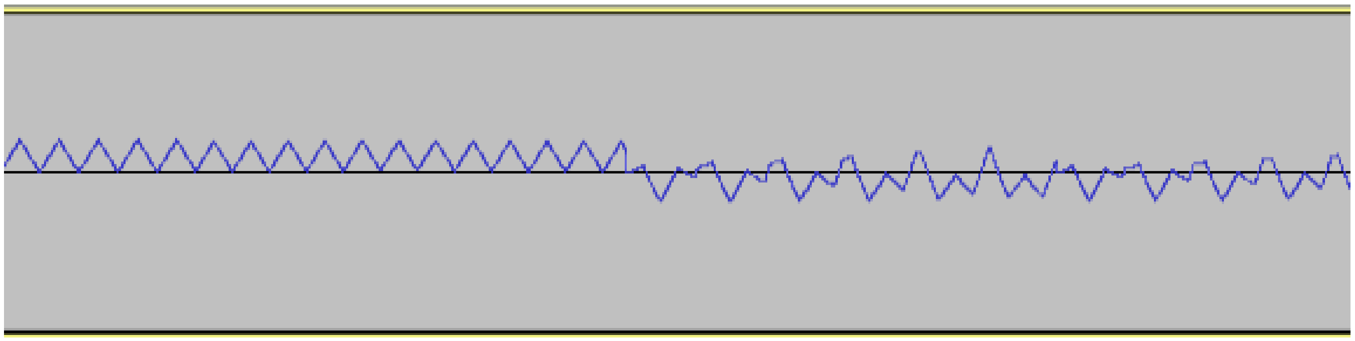

Figure 1: Sample output as viewed in Audacity

ABSTRACT

My goal with this project is to create a Synthesizer or wave form like effect from the detection of Aruco markers. This is accomplished by generating wave forms based off of detected Aruco marker Ids and then modulating those wave forms based on positional data gathered from the Aruco markers. The wave forms are generated and played as audio in real time and saved for replay in an audio file.

KEYWORDS

Aruco Marker, Wave form, Frequency, Phase

ACM Reference Format:
Alexander Tedrow. 2022. Aruco Marker Synthesizer. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Aurco marker detection has a wide range of applications in the world of computer vision. Most of these applications fall into the visual domain and use Aruco markers to manipulate some visual output. For my project I am taking a different approach by applying Aruco marker detection to the audio domain. By using Aruco marker detection to manipulate the output of audio I have created an Aruco marker synthesizer or wave form generator.

To detect the Aruco markers I am using the OpenCV library with python, and I am using the `SciPy`, `numpy`, `wave` and `simpleaudio` libraries for generating wave forms and outputting audio in real-time and writing to a .wav file.

2 ARUCO MARKER DETECTION

Before detecting any Aruco markers the camera intrinsics and distortion matrix must be obtained. I accomplish this by feeding the program a short calibration video of a calibration chessboard, and get the matrices needed from the first several frames of the calibration video.

After the camera intrinsics and distortion matrix are obtained the program can parse through each frame fed to it by the camera and detect any Aruco markers using OpenCV. The positional information of the Aruco markers can be gained from the corners object that is returned when calling `detectMarkers()` with the X and Y positions for each detected marker being given by corners[markerID][0][0][0] and corners[markerID][0][0][1] respectively. Once the positional data has been obtained and displayed it can be used to generate and modulate an audio wave.

3 WAVE FORM GENERATION

Once all Aruco markers are detected within a frame and I have obtained their positional information, I can use the marker's positional information to generate a unique wave form. The type of wave is determined by the Aruco markers ID with an ID of 0 producing a

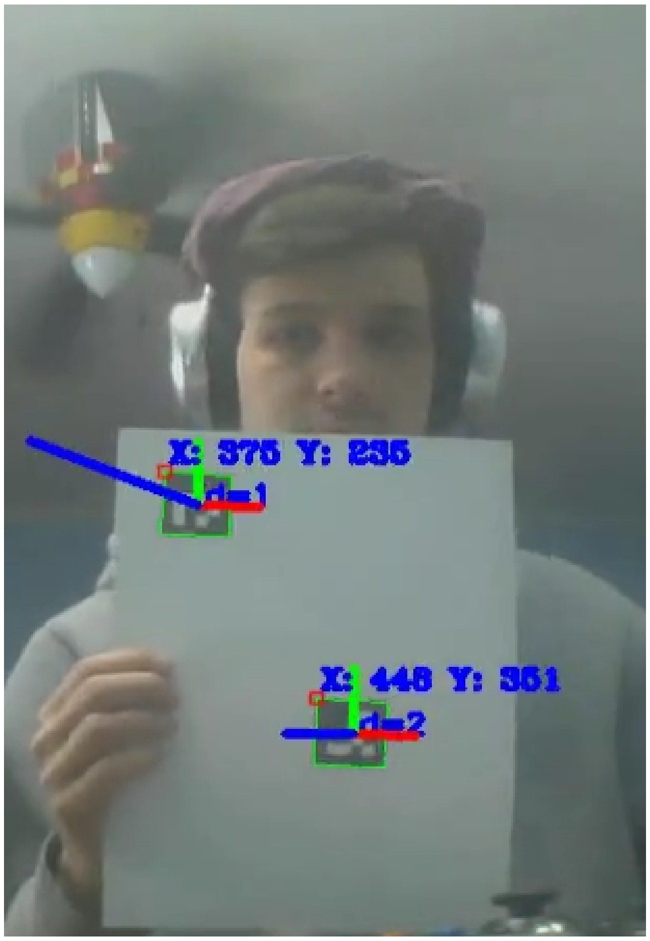

Figure 2: Example display of Aruco Marker Detection

sine wave, 1 producing a square wave, and 2 producing a sawtooth wave. However, all Aruco markers in the given dictionary are valid and will produce a wave form based off of:

```
wave_type_index = aruco_id % 3
```

Other aspects of the wave forms I alter based on the Aruco markers are the frequency and phase of the wave forms. The frequency is modulated based on the Y positioning of the marker and the phase is modulated based on the X positioning. Converting the X position data into a phase between $-\pi/2$ and $\pi/2$ is trivial since they both lie on linear scales. However, frequency is on a logarithmic scale which makes the conversion from the linear Y scale more complex. To convert X position data into the frequency domain I need to perform:

$$\min frequency \times \frac{maxfrequency}{minfrequency}^{\frac{index}{maxindex-1}}$$

Once I have obtained the frequency, phase, and wave type for each detected they can be passed to my wave generation method that will generate the wave and output it directly to audio. While each wave type is generated differently through SciPy or numpy the general form for the sinasoid equation goes as follows:

$$2\pi \times (frequency + phase) \times \frac{currentSegment}{samplerate}$$

Where the current segment is the current point in the duration of the waveform being generated, essentially the time variable t as it is usually represented in the sinasoid equation. This equation is then passed to its respective wave type function and the given value is added to the running value, so that multiple wave forms can be combined together in real time to produce a more complex wave form.

4 CONCLUSION

I consider this project to be successful as my program is able to generate complex wave forms from the detection of Aruco markers in real time. While detecting the Aruco markers in real time was fairly straight forward, generating the wave forms and actually outputting audio in real time proved to be much more of a challenge with Python3. I found that many popular audio libraries for Python2 have become depreciated with new versions of Python3. After trial and error with a few different libraries I found the `simpleaudio` library to be both up-to-date and functional. The library is limited in its functionality, but it allowed me to write a buffer of data to audio output and worked with .wav file data which was perfect for how I had set up my program.

There are some remaining issues with the program. Producing sawtooth waves is particularly slow as SciPy has to calculate several sinasoids to produce a sawtooth wave and this significantly slows down the program. Processing can be slow when there are several Aruco markers on screen as well, and this leads to poor audio output during real time operation of the program. This may be able to be fixed with multi-threading, however I have never attempted multi-threading with Python and felt it was outside of my scope for this project. There is also some slight audio issues with the wave generation. Not all the waves generated line up exactly with each frame so there is some slight stuttering on playback.