Visual to Auditory Sensory Conversion

Finn Boire[a)]

(Dated: August 21st, 2017)

The common challenge faced by those who are visually impaired is that they are unable to utilize to its fullest potential one of the most versatile and dynamic senses that is available to human beings: sight. Tactile and auditory inputs allow them to still maintain an awareness of the world, but it lacks accuracy and completeness that would be granted to them through a visual input. To resolve this issue, I created an apparatus that converts camera inputs to sound. The mappings all vary; from color values and depth coming from the cameras, to pitch, volume, and panning coming out of headphones. In this paper, I will discuss the merits and faults of the various mappings I attempted.

I. INITIAL ISSUES

A. Dimensional Problems

There are inherently several difficulties that I faced when trying to convert a 2-tensor of four dimensional vectors to that of a 1-tensor of three dimensional vectors, where the four dimensional vectors contain three color values and depth, while the three dimensional vectors contain frequency, panning, and direct volume. The majority of this paper will be devoted to this problem, examining various solutions and mappings to create an intuitive and easy to use interface.

B. Depth Detection

One aspect of human vision that is difficult to incorporate into an auditory device is depth. An ear cannot distinguish between two identical sounds coming from different distances away (provided volume is the same upon reaching the ear), and when all sound produced is through a pair of headphones, it is difficult to produce sound that seems "far away."

II. DEVELOPMENT OF THE APPLICATION

The application arose through an idea to create interactive audio landscapes through camera inputs. However, as I saw the potential for what camera to audio processing could do, it morphed into a wearable sensory supplement.

It's structured in two threads, running simultaneously. To reduce processing delays, there is the image processing thread and an audio generation thread, which send data between each other via protobuf messages. Within the vision thread, OpenCV 2.0[1] is used to manipulate the input from the webcams, then sends the resulting data through protobuf to the audio thread, which uses PortAudio[2] to generate sine waves of a particular frequency. Development occurred during the months between June and August, as did the testing of the various methods.

The largest hurdle faced was the experimentation with depth sensing capabilities. Not only was the substantial price of cameras a problem, but also the APIs. The prices associated with many of the depth sensing cameras on the market (be they time-of-flight, infrared, or stereo) were outside of my budget, and the APIs for many of the lower-end cameras that I could afford were badly documented and unreliable. As a result, I had to develop code to do some stereo vision processing. OpenCV already has several built-in functions to do something along the sort, but to make OpenCV's functions work, I needed to connect them to the previously existing code. I calibrated the cameras using a checkerboard pattern, but had difficulties in creating a precise depth measurement.

I used the depth data to change the volume of the audio sources depending on how far away they are, making it easier to tell of the proximity to an object. It's still not working 100%, but in tests where it has worked, the depth measuring contributed greatly to improving the ability to understand the audio, and tell what was either close or far away. A potential way to improve depth data and portability of the application would be to deploy to a depth-sensing smartphone, like a Google Pixel or iPhone 7, which doesn't use a jerry-rigged depth sensing setup like the current application.

The second largest issue encountered in programming the application was in accessing the camera streams. Due to insecure connections, I had a hardware issue when using one of the cameras, which caused the application to segmentation fault when trying to access pixels that held null values.

Third, incorporating OpenCV and PortAudio into the Bazel project took trial and error, serving as a good learning experience for incorporating third-party programs into a C++ application built with Bazel. Ultimately, after unsuccessfully adding a third_party folder which included the source for both libraries and custom rules for including the libraries, I tried a global installation of both OpenCV and PortAudio on the system. This allowed us to include symbolically linked files to the compiled libraries, instead of compiling themselves as a part of the

a)http://steampunc.com

projects.

These were the main difficulties faced in the development of the application. The rest of the paper will be dedicated to addressing the potential input modes and their merits.

III. VISUAL TO AUDITORY MAPPINGS

A. Colorspaces

1. RGB

The most commonly known and well-understood form of storing colors within a computer's memory is through the RGB colorspace, where three values, corresponding to a level of red, green, and blue respectively, vary to describe the range of possible colors. This colorspace is helpful from a computer programming standpoint, because computer displays use red, green, and blue pixels to display content, but it's not as useful for image processing, because it is difficult to find correlation between colors. For that reason, the RGB colorspace was avoided in this application.

2. HSV

A HSV colorspace holds three values, much like the RGB colorspace. However, these values, instead of holding direct correlation to the amount of a given color, are instead controlling the hue, saturation (or brightness), and value of a colored pixel.

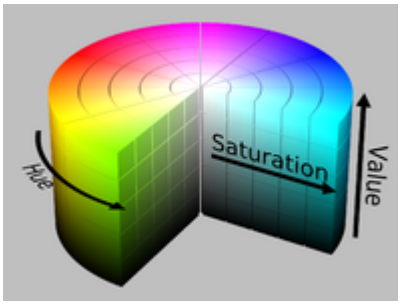

FIG. 1. The HSV Colorspace

Fig. 1 shows the variations in each value and how it affects the color displayed. As saturation nears zero, it becomes more white, and as value increases it gets less dark and move vibrant.

This colorspace is useful because it gives a single value that controls the range of colors. This means that in programming a direct correlation between pitch and color, the hue value is linearly linked to pitch, only scaling by a constant. In tests, this pitch linking has shown to be a fairly successful way to determine the color of an object without a visual feedback. Taking advantage of neuroplasticity, over time the brain develops an understanding of what each pitch means, even for those who lack perfect or relative pitch abilities, as I saw in other studies[3]. Although not yet reaching such results with the current application, there is potential to get to this further and more advanced stage.

The second potential benefit of the HSV colorspace is the value level. By mapping the value to volume, the more vibrant the color, the louder the sound of this color. This allows the user to determine subtleties in the color and brightness of their surroundings that would have gone unnoticed without some sort of feedback. However, although providing nuance to the auditory input, this value-based modulation came with pitfalls in testing. With too many variables controlling volume, it was difficult to interpret what a variation in volume meant. Any object could have gotten further away (as depth also controls volume), darker (due to the value changing), and with the addition of other variables, there could be even more affecting it. For that reason, in the final version, there was no value controlling volume or pitch.

3. Greyscale

Grayscale was the other colorspace that saw experimentation. This colorspace only holds one value per pixel: its brightness. With this more simple input, I mapped the brightness to pitch. In tests, the simplicity of this colorspace made it the easiest to begin interpreting. After wearing the apparatus for a prolonged period of time, testers were able to distinguish between different parts of a room and outdoor areas. However, this mapping seemed to make it more difficult to determine specifics about surroundings, although it simplified generalized recognition.

B. Audio Panning

One aspect of auditory sense not yet discussed in this paper is the ability to determine lateral differences between sounds. We can tell the difference between a sound coming from the left and the right, so for all mappings, the further to the left that the pixel is situated in the camera input matrix, the more volume coming from the left side of the headphone set. Within the code, the pixel's coordinates in the input matrix takes the form (y, x), where x increases the further to the right the pixel is positioned. Thus, the corresponding volumes are:

$$\mathrm{V_{Left}} = \frac{\mathrm{image_{width}} - x}{\mathrm{image_{width}}}$$

$$\mathrm{V_{Right}} = \frac{x}{\mathrm{image_{width}}}$$

With these calculations, I was able to make the pixel's position affect the volume in both ears, giving the sensation of a directional sound.

This directional sound is imperfect. For our brains to understand a true directional sound in an environment, our ears must perform some subconscious calculations, accommodating for the speed of sound and the minute delay between reaching one eardrum and the other, and the slight muffling that occurs depending on the direction of the sound. The sine waves coming from the headphones don't simulate this, and it is another part of the application that needs further development.

C. Grouping

The final part of the program addressed in this paper is the grouping of pixels to produce sound. If the application was generating an individual sine wave for each and every pixel of the camera input matrix, it would not only take significantly more processing time, but would also create such a cacophony of noise that, without extensive acclimation to the system, prospective users would face difficulty interpreting inputs. So, to simplify the interface, the program divides the camera input matrix into a customizeable number of groups. First, the program blurs the image, then samples a single pixel from each group and sends the data for that pixel through a protobuf message to the audio generation thread.

IV. CONCLUSION

The (only partially up-to-date) code for the project is on Github[4], but you can see many of the things written of in this paper there. In the final iteration, although details were still not easily interpretable, the program allowed users to navigate through hallways, along sidewalks, and go about their lives. The current barriers to progress are portability, consistent depth measurement, and price (I don't have infinite money to spend on this project), but with more time and effort directed towards these barriers, they will be solved. In moving forwards, I am considering the usage of a phone for processing and an industrial-grade depth camera, optimizing code for a slightly less powerful computer, and ultimately establishing a product for production within the next year.

[1]Http://opencv.org/.
[2]Http://portaudio.com/.
[3]I have misplaced the link to these studies. I assure you they existed on the internet in June of 2017.
[4]Https://github.com/steampunc/blind-helper.