

---

---

LISTENING TO LIGHT  
MAJOR QUALIFYING PROJECT

---

---

MAY 2016

AMANDA VARRICHIONE

PHYSICS

RYAN LANG

PHYSICS

*Advised by*

RICHARD QUIMBY

PROFESSOR OF PHYSICS



## **Abstract**

A color detecting device was designed and implemented by two students at Worcester Polytechnic Institute to satisfy their Major Qualifying Project requirement for obtaining a Bachelor of Science in Physics. This device could distinguish ten different colors, and could correct for background lighting intensity and distribution changes. Based on the detected color, the device would play a unique three-note sequence, audially indicating the detected color. The device could accurately detect the color of a sheet of 8.5" by 11" card-stock from a distance of 35 cm for all colors and background sources.

## **Acknowledgments**

We would like to thank our project advisor, Professor Richard Quimby for the help and guidance in making this project. Additionally, we would like to thank Russell Lang for his advice and help in designing and implementing the prototype.

We would like to thank the Physics and Civil Engineering departments at WPI for allowing us to use their facilities, workshops, and laboratories.

Finally, we would like to thank Tony DiCola, whose open source Python library for the Adafruit TCS34725 color sensor was vital to the implementation and completion of this project.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Background</b>	<b>6</b>
2.1	Color Theory . . . . .	6
2.1.1	The Visible Spectrum . . . . .	6
2.1.2	Human Color Perception . . . . .	6
2.1.3	Combinations of Color . . . . .	8
2.1.4	Spectral Distribution of Common Lighting Sources . . . . .	8
2.2	Optics . . . . .	9
2.2.1	Optical Fibers . . . . .	9
2.2.2	Filters . . . . .	10
2.2.3	Photodetectors . . . . .	11
2.2.4	Apertures . . . . .	12
2.3	Audio . . . . .	13
2.3.1	The Science of Sound . . . . .	13
2.3.2	Human Sound Perception . . . . .	13
<b>3</b>	<b>Methods</b>	<b>15</b>
3.1	Approach . . . . .	15
3.1.1	Raspberry Pi . . . . .	15
3.1.2	TCS34725 Color Sensor . . . . .	16
3.2	Fiber . . . . .	22
3.2.1	Selecting a Fiber and Numerical Aperture . . . . .	22
3.2.2	Core-Cladding Structure of the Fiber . . . . .	24
3.2.3	Sensor to Fiber Alignment . . . . .	27
3.3	Aperture . . . . .	30
3.4	Device . . . . .	32
3.4.1	Device Components . . . . .	32
3.4.2	Device Layout . . . . .	33
3.5	Python Program . . . . .	34
3.5.1	Obtaining and Manipulating RGBC Values . . . . .	35
3.5.2	Account for Background Lighting Distributions . . . . .	36
3.5.3	Learn Colors and Backgrounds . . . . .	36
3.5.4	Identify a Color . . . . .	37
3.6	Audio Output . . . . .	38

<b>4</b>	<b>Results and Conclusions</b>	<b>40</b>
4.1	Accuracy . . . . .	40
4.2	Limitations . . . . .	41
4.3	Conclusions and Future Work . . . . .	41
<b>5</b>	<b>Appendices</b>	<b>43</b>
5.1	Python Code . . . . .	43
5.2	Device Operation . . . . .	54
	<b>References</b>	<b>56</b>

# 1 Introduction

Color is all around us. Every day, we experience color from the moment we open our eyes to the moment we close them. It is a vital part of human vision, and we use and rely on it in many facets of our lives. The color of road signs and stoplights gives a driver immediate information in a universal medium. One can drive safely in foreign lands relying only on color cues for instructions. It also plays a huge part in aesthetic applications, where artists and designers use the millions of discernible colors to appeal to our eyes. The idea of colors as we experience them is truly something unique to the human experience.

Colors, however, only exist in the human mind. They are a construct of the brain, which creates colors in our conscious mind as a representation of what started as different light wavelengths. Those without the ability to experience colors are unfortunately denied a unique part of the human experience. There have naturally been many attempts to design and implement devices that could discern the colors of objects, usually reporting them in some kind of audio format. Some recent examples are the Adafruit PianoGlove and the different versions of the Eyeborg. This kind of device is naturally limited by the bandwidths of the human visual and audial sensory systems. The visual system typically transmits data at a much greater rate than the audial system. Therefore, transmitting visual information using sound waves is fundamentally limited. Still, devices that detect color and use sound to communicate those colors have been successfully implemented and are worth pursuing.

The previous implementations have had some significant shortcomings, however. The first is the issue of background lighting distributions. The lighting sources in our everyday lives have wildly different spectral distributions, which can wreak havoc for a color detector. The human eye naturally adapts to different intensities and changes in spectral distributions. Objects look the same color in differing sources because of this, but to an artificial sensor these differences will completely change the data obtained. Secondly, previous iterations have used single notes to represent different colors. While it is easy to assign many different note-color combinations this way, the human ear is usually very poor at distinguishing absolute pitch (unless one has 'perfect pitch'). This makes these previous systems unintuitive and difficult to use, especially for a colorblind or blind person with no other way to distinguish colors.

The overall mission of this project was to improve upon similar devices by attempting to remedy their weaknesses. The group aimed to design and implement a similar device that would be able to adapt to changes in background distribution, and in doing so create a more eye-like version of these devices. The group also wanted to output audial information in a format that was more intuitive and utilized the natural strengths of the human audial sensory system. In doing so, the group hoped to create a prototype to demonstrate the feasibility of these improvements for a device that could assist the visually impaired.

## 2 Background

The background is intended to give the reader the necessary information to understand the methods used to carry out this project. This section details research about color theory and how the eye sees color, the study of optics and optical devices, and the theory of sound and human audio perception. This research led to significant design implications for the implementation discussed in the next section.

### 2.1 Color Theory

#### 2.1.1 The Visible Spectrum

Color, as humans see it, is comprised of a range of wavelengths in the electromagnetic spectrum. This range of wavelengths is known as the visible spectrum as it is visible to the human eye. The visible spectrum ranges from about 380 nm to 780 nm.[28] Figure 1 below shows the visible spectrum as a part of the whole electromagnetic spectrum.

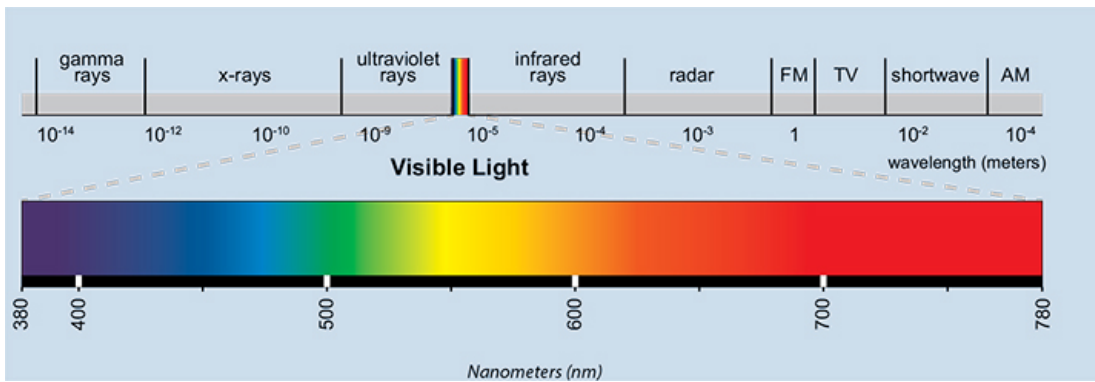


Figure 1: The visible light spectrum [9]

Figure 1 illustrates that as the wavelength decreases from 700 nm to 400 nm, the corresponding color at each wavelength progresses from red to violet. Wavelength ranges are assigned to different colors to represent how light of a certain wavelength is perceived by the human eye.[28] The frequency of a light wave is inversely proportional to the wavelength by the following relation:

$$\nu = c/\lambda \quad (1)$$

This relation allows us to determine the color of a light wave with its given frequency.[30]<sup>1</sup>

#### 2.1.2 Human Color Perception

Humans perceive color with special cells in the eye that have different spectral sensitivities. These are known as cone cells. Humans have three types of cone cells giving us trichromatic color vision.

---

<sup>1</sup>c = the speed of light in a vacuum

This means every color that we can perceive is determined by measuring the energy levels in three spectral bands and processing that information. The three types of cone cells, known as S, M and L, respectively correspond to a 420-440 nm band, a 534-550 nm band, and a 564–580 nm band.[29] Each type of cone cell has a variation of the protein photopsin, which determines the cone cell’s spectral responsivity. It can be helpful to think of these cones types as blue (S), green (M) and red (L) bands, although they don’t exactly match these colors. The exact spectral responsivities of the S, M and L cones can be seen in Figure 2. In addition to the cone cells, the eye uses the help of rod cells to detect blue-green light, while the M and L cones can detect subtle differences in green, yellow, orange and red hues because their response bands are very close together.

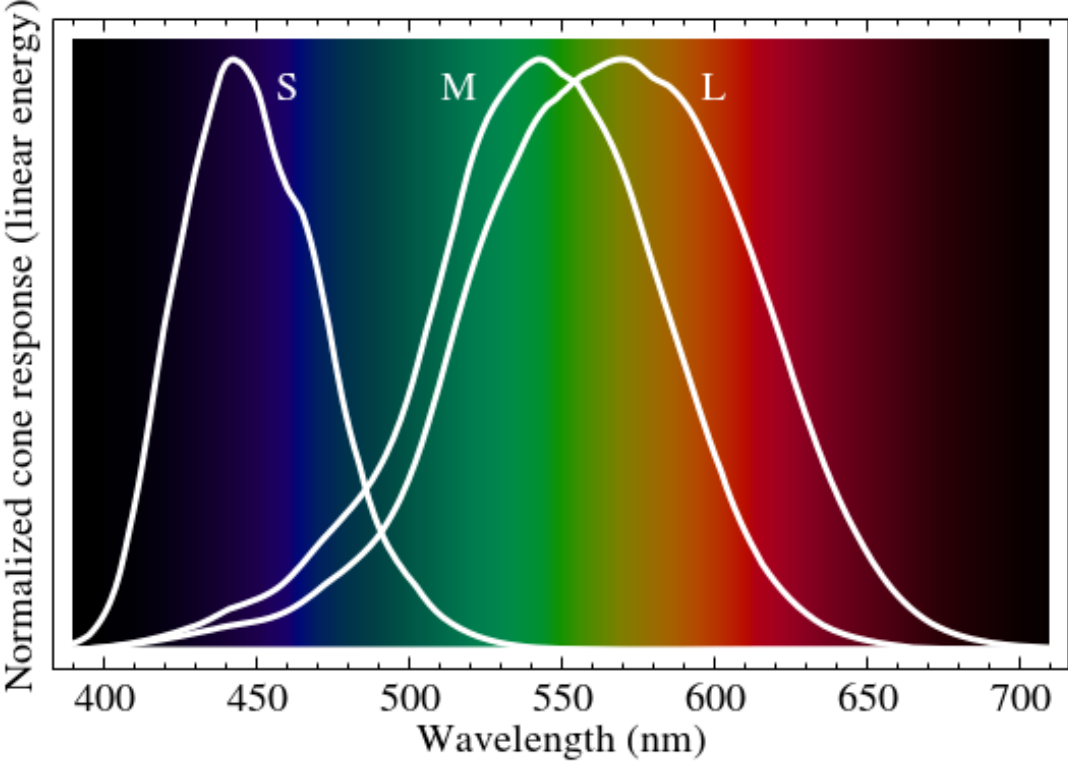


Figure 2: The spectral sensitivity of S, M and L cone cells

The prevailing theory about color processing in the brain is the trichromatic theory, or the Young-Helmholtz theory. According to this theory, the three cones being preferentially sensitive to red, green and blue light allows the brain to distinguish between millions of hues.[31] For example, we ‘see’ cyan when our S and M cones are partially excited, while the L is less excited, even though there is no cone tuned in to directly measure the cyan wavelength. By knowing the excitation levels for each cone, the brain is able to distinguish around 10 million different colors.



### 2.1.3 Combinations of Color

As stated previously, the human eye only detects wavelengths in three bandwidths which lie approximately in the red, green, and blue ranges. However, humans still sense all colors in the visible spectrum because the brain processes combinations of red, green and blue into new colors. The secondary layer of color past red, green and blue consists of magenta, yellow, and cyan. Figure 3 shows the secondary colors created from combinations of the primary colors.

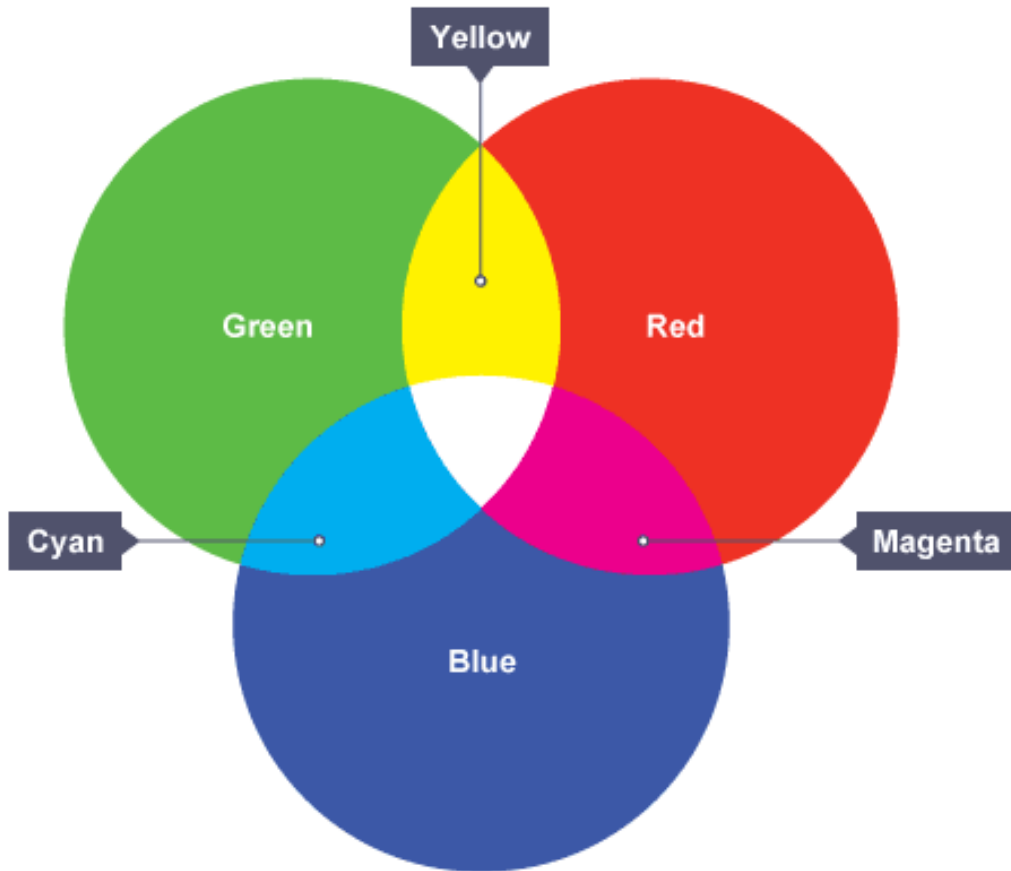


Figure 3: Primary color combinations [16]

From Figure 3 it can also be seen that the combination of red, green, and blue altogether creates white while the absence of any color creates black.

The colors red, green, and blue are known as the primary colors of the eye. Primary colors are any set of three colors that, when combined in different proportions, can create any color in the visible spectrum. Another example of a set of primary colors is the combination of cyan, magenta, and yellow which is commonly used as the three ink colors in printers.[16]

### 2.1.4 Spectral Distribution of Common Lighting Sources

One of the main differences between commonly found lighting sources is the spectral power distribution (SPD) that each bulb emits. A spectral power distribution is a representation of the intensities

of light at each wavelength over the visible spectrum. The SPD of some bulbs, like natural sunlight, is relatively continuous meaning it contains a nearly equal amount of light for every wavelength in the spectrum. Other light sources, like a compact fluorescent lamp (CFL), are not continuous.[25] A CFL bulb only contains spikes of intensity around three wavelengths: one in the red range, one in the green range, and one in the blue range. The spectral power distribution of sunlight, a CFL bulb, and a tungsten lamp is shown in Figure 4.

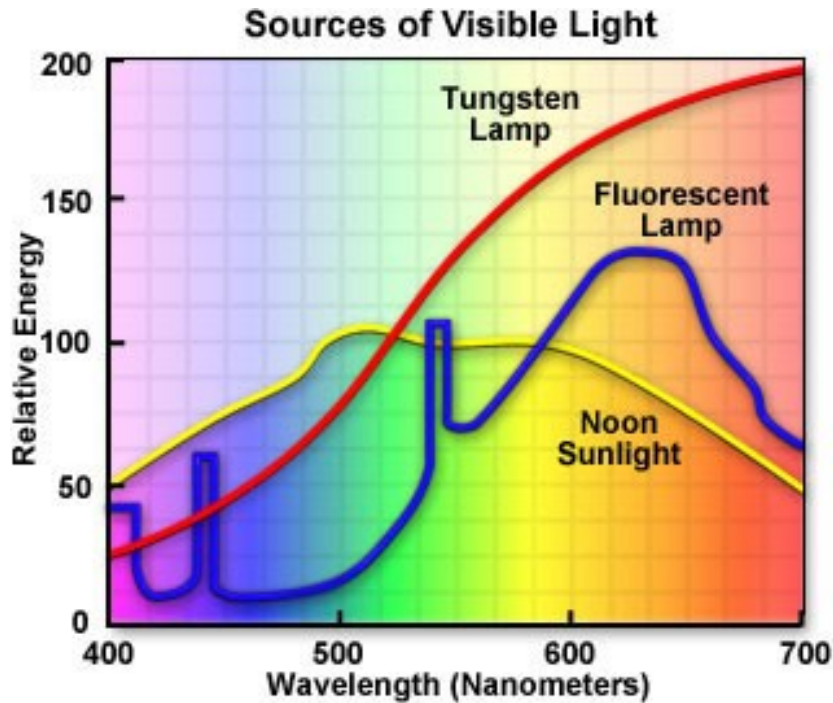


Figure 4: A spectral power distribution of Sunlight, a CFL, and a Tungsten Lamp [6]

The figure illustrates that tungsten lamp has a relatively continuous spectra like sunlight, while the CFL bulb is discontinuous. Although the SPD of these light sources are different, they both output what appears to be white light. Therefore, in order to determine the color of an object, one must first consider the spectrum of the background lighting source before considering the spectrum of the illuminated object.

## 2.2 Optics

### 2.2.1 Optical Fibers

Optical fibers are used as a means to transmit light as information along the fiber strand. Optical fibers are generally made of thin strands of silica glass but can also be made of plastic. Fibers can be used for many applications, however in this project we are only concerned with fibers being used to transmit visible light.[19]

Light travels in a fiber by total internal reflection as seen in Figure 5 below. Light in a fiber is reflected at the interface between the core and the cladding of the fiber. Light incident on the

fiber may escape through the cladding depending on the angle at which the light beam hits the core-cladding interface. Ideally, light travels through the entire fiber without escaping until it reaches the end of the fiber.

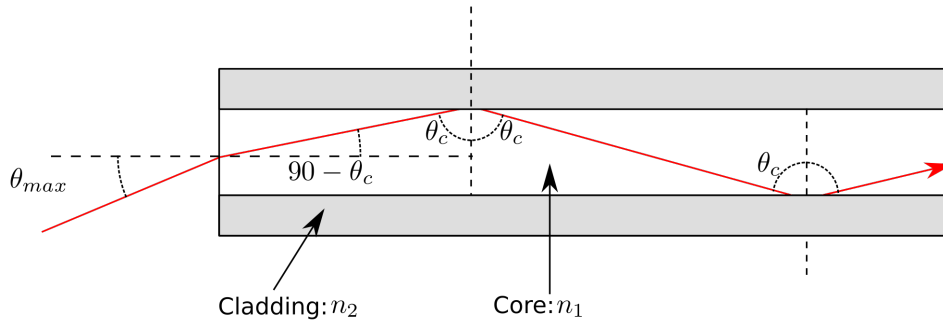


Figure 5: Inside an optical fiber [19]

The core and cladding of the fiber are made from different indices of refraction with the cladding index lower than the core's. The critical angle that determines whether or not light inside the core will be reflected or transmitted at the core-cladding interface is given by the equation:

$$\sin(\theta_c) = n_2/n_1 \quad (2)$$

If the light beam hits the core-cladding at an angle greater than the critical angle (with respect to the normal line between the interface), then the beam will be transmitted through the core-cladding interface instead of being reflected.

The numerical aperture of a fiber is an inherent characteristic that gives the maximum acceptance angle for light to enter that fiber. The numerical aperture (NA) is given by the following relations:

$$NA = n_o \sin(\theta_{max}) \quad (3)$$

$$NA = \sqrt{n_1^2 - n_2^2} \quad (4)$$

Here,  $n_o$  is the refractive index of the medium outside the fiber,  $n_1$  is the index inside the fiber core, and  $n_2$  is the index inside the cladding. The numerical aperture of a fiber is typically given by the fiber manufacturer. As seen in equation 3, a larger NA will give a larger angle of acceptance which could be beneficial depending on the fiber's function.[21]

### 2.2.2 Filters

An optical filter is device used to limit light transmitted through the filter area to a selected range of wavelengths. Filters come in two types: interference filters (also called dichroic filters) and absorptive filters. Absorptive filters absorb the wavelengths of light that do not pass through the filter while interference filters reflect these wavelengths.

Bandpass filters are a certain type of optical filter that allows transmission only for a band or range of wavelengths and blocks all other light. These filters are often categorized by which color range of the visible spectrum the transmitted wavelengths fall in.[26] Figure 6 gives the transmission curve for a green dichroic filter from Thor Labs.

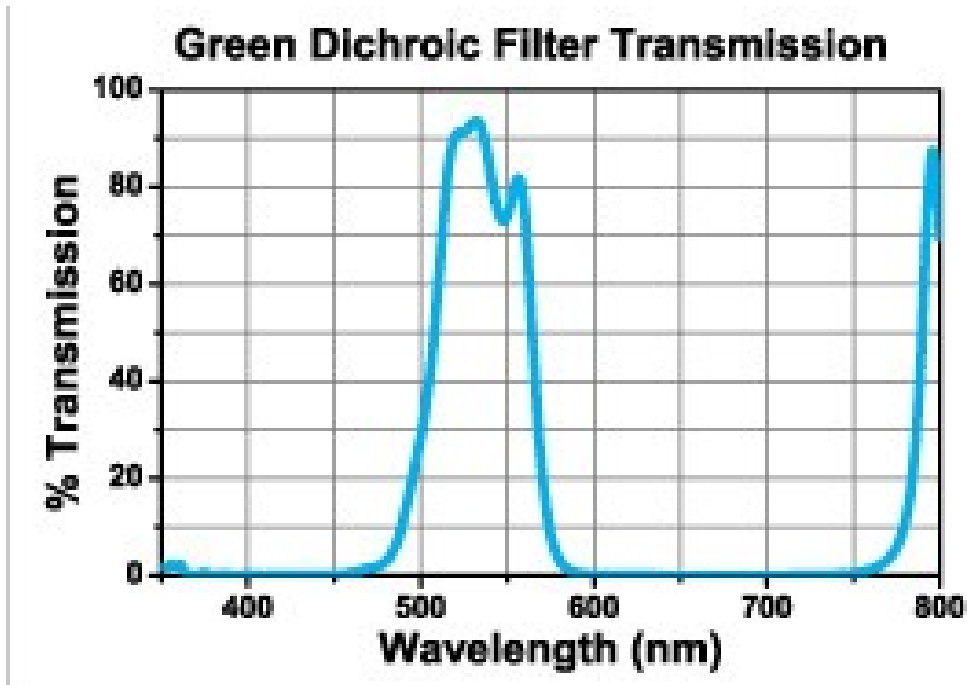


Figure 6: Transmission graph for a green dichroic filter [8]

Figure 6 illustrates that the green dichroic filter only allows wavelengths from about 475 nm to 575 nm to pass through the filter. Additionally, we can see that some infrared light is allowed to be transmitted through this filter. In order to block IR light from being transmitted, some optical filters are coated with an IR blocking film.[8]

Interference filters, or dichroic filters, work by the principle of interference. Wavelengths that do not pass through the filter interfere destructively to cancel each other or are reflected in the medium when the waves overlap.[26]

### 2.2.3 Photodetectors

A photodetector is a device which can effectively measure the intensity of light incident on an active area. Most photodetectors today are silicon based, and translate the amount of incident photons on an active area into a voltage. Therefore, by observing the voltage, one can effectively measure the intensity of light, assuming a linear relationship between light intensity and voltage levels produced.[21]

Photodetectors have a spectral sensitivity that describes how sensitive the detector is to different

wavelengths of light. This sensitivity can be altered by placing a filter in front of the detector. In this way, a detector can be made to only be sensitive to a prescribed range of wavelengths. Photodetectors can be made to be very flexible; for example, the sensitivity (to all light) can be tuned by adjusting the bias voltage, which also changes the response time [21]. The dynamic range of a photodetector is a term that describes the ratio between the lowest and highest intensities of light that can be distinguished, and this number can be quite large. A large dynamic range implies that a photodetector could be used under a variety of lighting conditions and under vastly differing intensities.

### 2.2.4 Apertures

An aperture is a rather simple device that is used to constrict the width of a beam of light. If one looks down the end of the cardboard tube from a paper towel tube, one has created a version of an aperture. When used in front of some kind of photodetector, an aperture tends to limit the angular acceptance of light incident of the photodetector. Typically, detectors could accept nearly any angled beam, but an aperture narrows that to allow greater precision at the cost of intensity (and perhaps signal to noise ratio). An aperture can also be used to limit the acceptance angle of an optical fiber. With an aperture in front of it, a fiber will only accept light from a further limited range of angles. As long as the modified acceptance angle created by the geometry and placement of the fiber is less than the acceptance angle for the fiber itself, all light that can pass through the aperture will be coupled into the fiber. Figure 7 shows the geometry of a typical aperture limiting the acceptance angle for a point (perhaps a photodetector or a fiber) on the left.

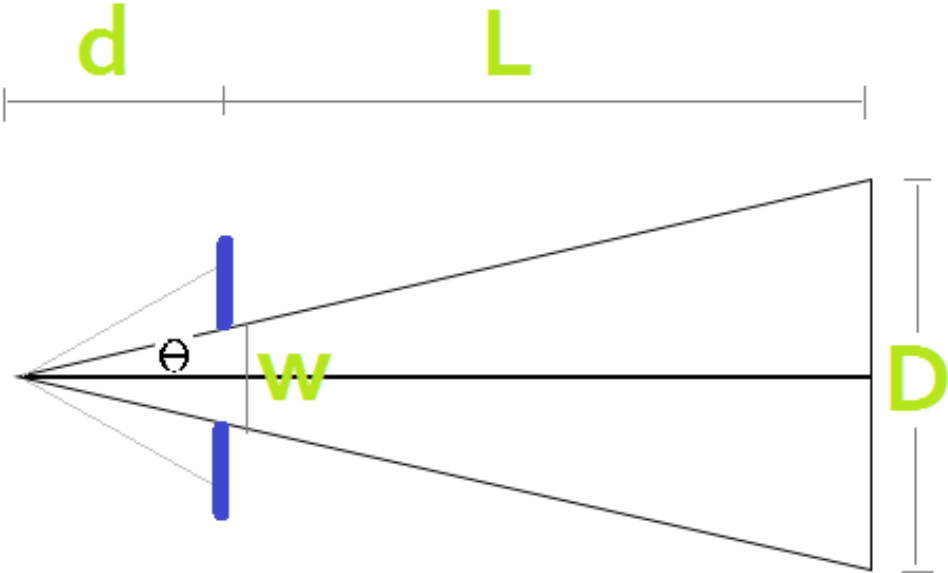


Figure 7: Geometry of an aperture

The relationship between the half angle of acceptance  $\theta$  and the determining factors: the distance from the aperture to the fiber opening ( $d$ ), the aperture diameter ( $w$ ), the distance from the aperture to the object ( $L$ ) and the spot diameter that the fiber will see ( $D$ ) is

$$\tan(\theta) = \frac{w}{2d} = \frac{D}{2(d+L)} \approx \frac{D}{2L} \quad (5)$$

The approximation in equation 5 is true when  $L \gg d$

## 2.3 Audio

### 2.3.1 The Science of Sound

Sound is a form of vibration that propagates through a medium. In air, it's a mechanical wave that is formed by local compressions and rarefactions of air molecules, as shown in Figure 8. These travel through the air at a nearly constant speed known as the speed of sound, about 343 m/s.[12] Due to the nature of the waves, the speed of sound is affected by the local air pressure and temperature. Sound waves have a characteristic frequency and amplitude. The frequency determines what we think of as the pitch. Higher frequency means that a higher pitch is perceived. Amplitude translates to loudness. A sound wave with a larger amplitude has more energy and is perceived as louder.

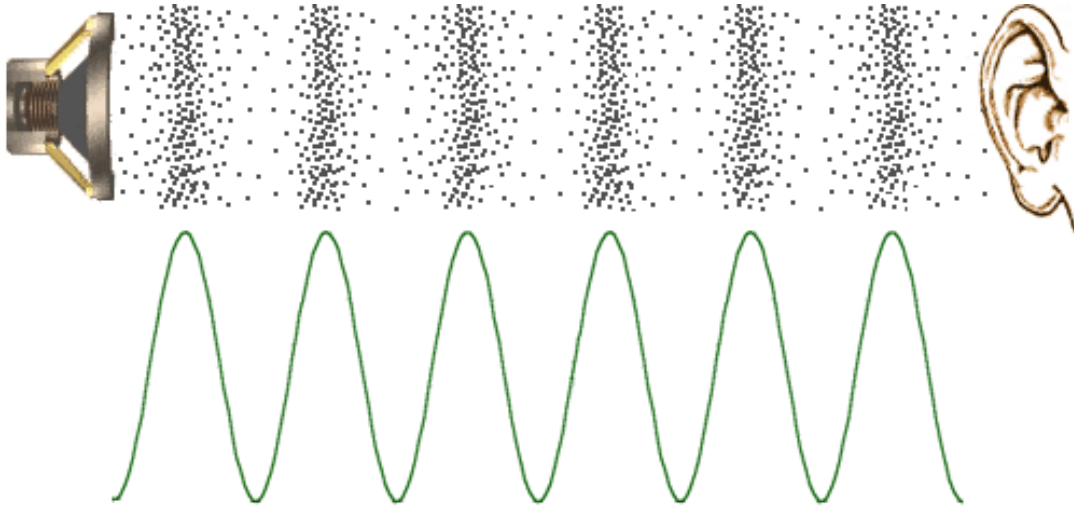


Figure 8: Variations in air pressure and corresponding waveform [14]

### 2.3.2 Human Sound Perception

The human ear can sense pitches from 20 Hz to nearly 20,000 Hz.[12] However, the ear's ability to determine absolute pitch is usually rather poor. It is considerably better at determining relative pitches, or how two pitches compare to each other. Specifically, our ear is adept at determining the ratio between two frequencies. It can identify the difference of 200 Hz to 400 Hz and 600 Hz to 1200 Hz as the same *musical interval* with remarkable accuracy. This relationship of a 1:2 ratio in

frequencies is called an octave in western music.[18] There are 12 intervals in western music, which correspond to the 12 possible distances between different piano notes within an octave. The ear can also easily distinguish relationships of notes when they are played sequentially, creating the notion of a melody. By relying on the strengths of the human ear, music can be used to convey information effectively and universally to any listener.

## 3 Methods

The methods section deals with the details of the actual implementation of our project. We discuss the components used in the final design, and the extensive tests and experiments done to discover their attributes. This section displays how our design choices were motivated by experimental results, and how the final device was designed to meet our objectives; to design, create, and demonstrate a universal and adaptable device to assist the visually impaired in distinguishing colors.

### 3.1 Approach

The methods section of our project is structured by analyzing the answers to two main questions: how can we detect colors in the form of light waves, and how can we output colors as sound? To detect colors, we decided to model our device after the inherent color perception system in the human body. As discussed earlier, human color perception is centered around the eye as a receptor of light and the brain as an information processor. The brain of our project is the Raspberry Pi, a portable microcomputer. The Raspberry Pi's function is to collect information from the 'eye' of our device, the TCS34725 color sensor. The TCS34725 color sensor works very similarly to the human eye as will be discussed in the upcoming sections. To output colors as sounds, we referred to human sound perception, paying special attention to the idea that melodies and chords are far more recognizable than single notes.

#### 3.1.1 Raspberry Pi

The Raspberry Pi 2 Model B, shown in Figure 9 has all the usual components of a desktop computer. With a 900 MHz quad-core ARM Cortex-A7 CPU and 1 GB of RAM, it is powerful enough for functions like web browsing and word processing. Powered by a 5 Volt micro-USB plugin, it has 4 USB ports for external devices, an HDMI port for a monitor, a headphone jack for audio, an Ethernet port, and 40 General Purpose Input Output (GPIO) pins.[22]

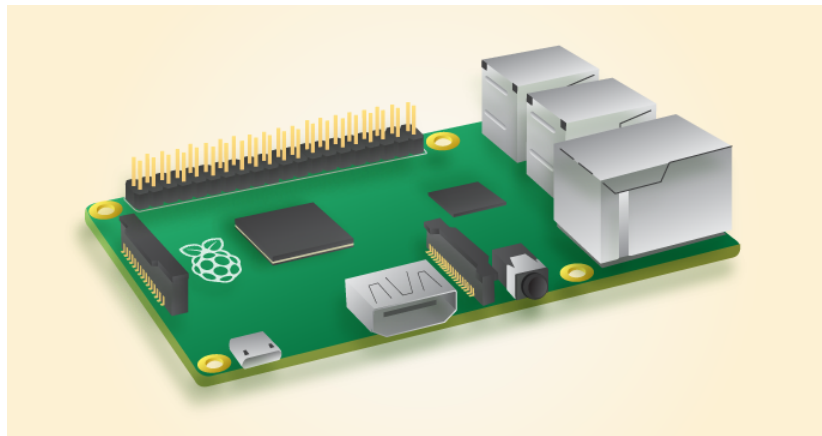


Figure 9: The Raspberry Pi 2 Model B [22]



We used the Raspberry Pi’s GPIO pins to connect to a color sensor that measures the intensity of light. As shown in in Figure 10, there are 26 available GPIO pins that can measure voltage change, shown as green pins. The Pi also has several ground pins and some voltage supply pins. Since the Raspberry Pi has a Linux-based operating system, we run a program from the root directory to have permission to access these pins.

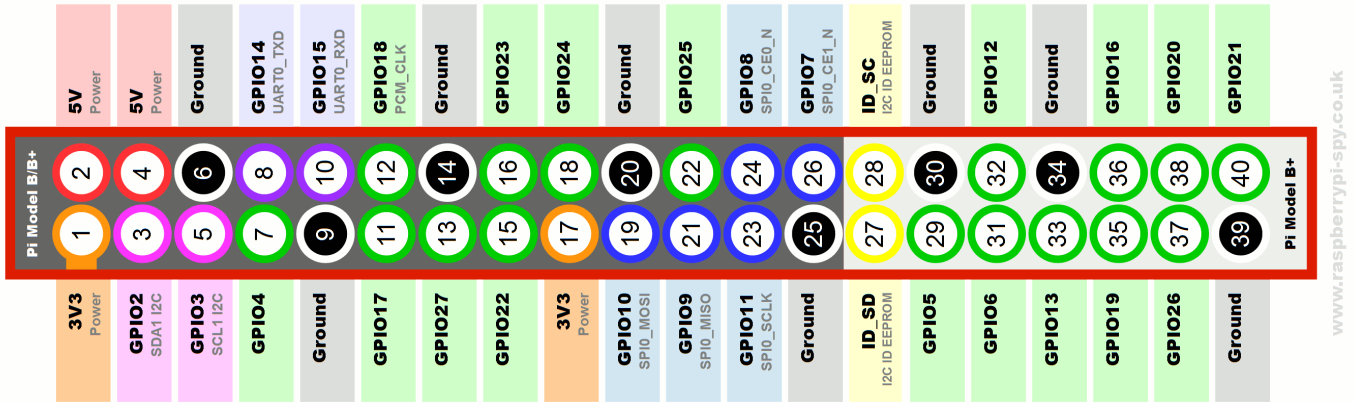


Figure 10: Schematic Diagram of the Raspberry Pi’s GPIO

We used the GPIO pins configured for i2c communication. These are pins 3 and 5; as seen in Figure 10, these purple labeled pins are accessed through the addresses GPIO2 and GPIO3. GPIO2 is described as SDA1 I2C and GPIO3 is described as SCL1 I2C. These correspond to the SDA and SCL connections in Figure 13.<sup>2</sup>

Although the Raspberry Pi can run most operating systems, a Debian based Linux distribution called Raspbian was written to provide an optimized operating system for general use. Raspbian is set up to be conducive to programming; it comes with Python, Scratch and other languages already installed as well as giving easy access to all of the Pi’s GPIO pins and other connections as long as programs are run with root permissions.

### 3.1.2 TCS34725 Color Sensor

The TCS34275 color sensor, shown in Figure 11 contains a 3x4 array composed of red, green, blue and clear filtered photodiodes that produce a voltage related to incident light intensity. All photodiodes are coated with an IR-blocking filter, and are designed to be responsive to red, green, or blue wavelengths of light, while blocking other wavelengths from reaching the photodiode.

<sup>2</sup>The specific connections set up with the GPIO pins is further discussed in Section 3.4

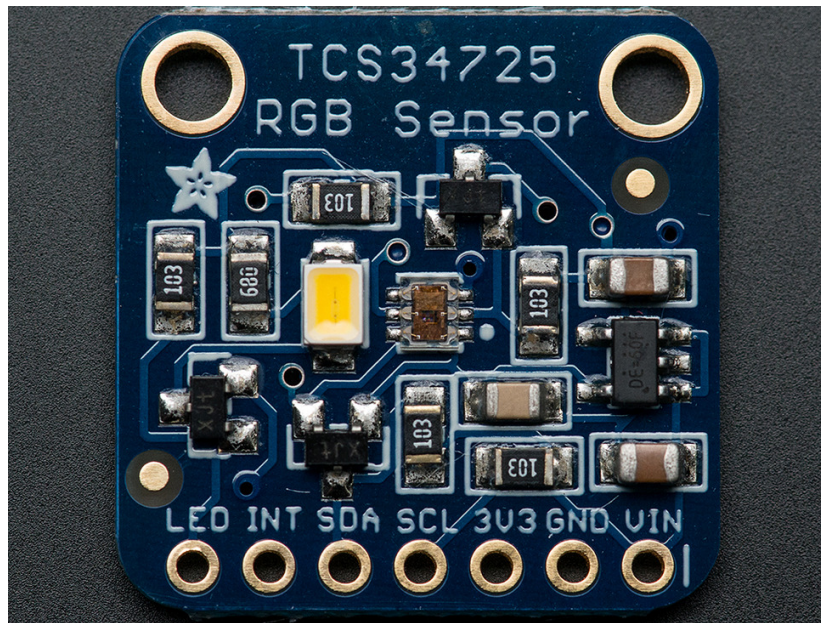


Figure 11: Image of TCS34275 device

The team used this device to simulate human color perception. It has the ability to report back effective values for red, green and blue intensities, which is very similar to the way the eye uses the three types of cone cells. For comparison, we have included the spectral response of the TCS sensor side by side with the previous figure showing the spectral response of cone cells in the human eye. Figure 12 below shows this comparison.

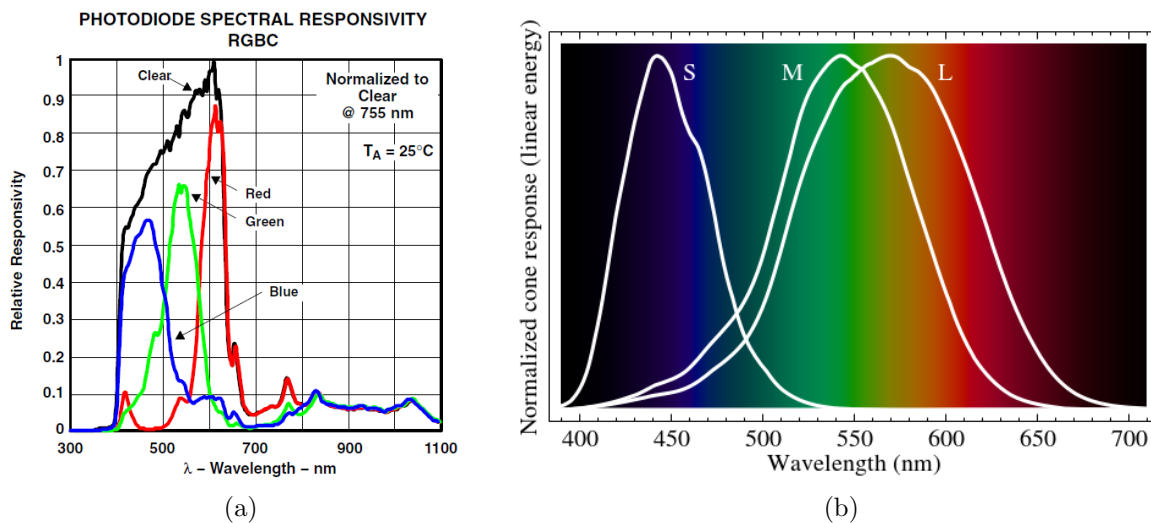


Figure 12: Left: TCS sensor spectral sensitivity [24]. Right: cone cell spectral sensitivity

There is a built in analog-to-digital converter in the TCS34275 which takes the analog voltages from the photodiode array and converts them into a industry standard digital communication format called i2c. A block diagram showing the information flow is shown in Figure 13 below. This format is a

400kHz, two wire serial bus used many communication connections to microcontrollers and embedded processors. The Raspberry Pi has two GPIO pins already configured to serve as an i2c communication bus, making communication with the TCS device relatively simple.

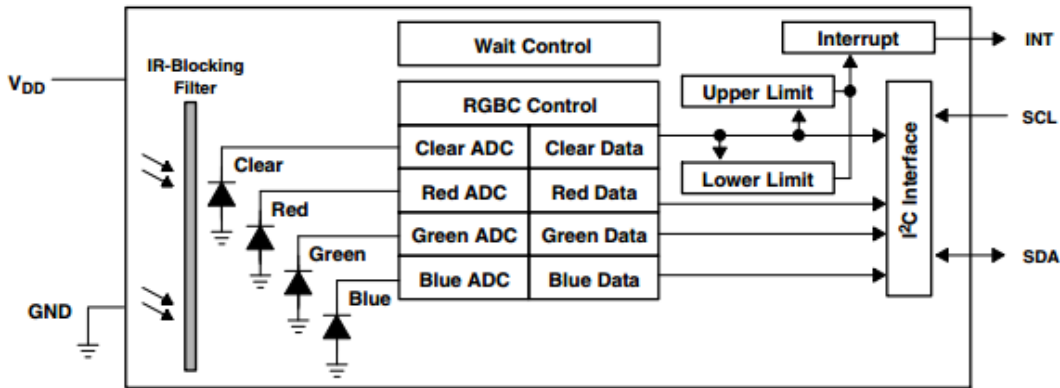


Figure 13: Block diagram for the TCS34275 device [24]

Connecting the sensor is fairly simple when following the proper instructions. The TCS34725 color sensor requires 4 ports to connect to the Raspberry Pi. The ports we used to connect the sensor are ports 1, 3, 5, 6, and 9. The ports and their purposes are labeled below in Figure 14.

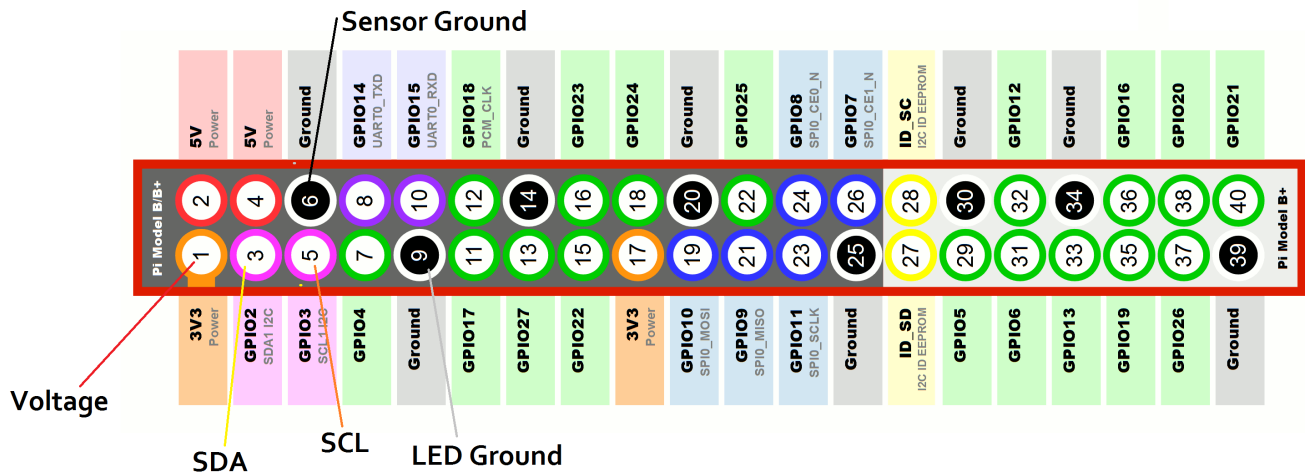


Figure 14: Raspberry Pi GPIO pins used by the TCS34725 color sensor [22]

The sensor was connected to the Raspberry Pi's GPIO pins using thin wires called breadboard jumper wires. The jumper wires are small enough so that the individual GPIO pins on the Raspberry Pi are easily accessible, and the wires help to consolidate the device design making it as portable as possible.

The TCS34725 color sensor itself contains seven pins for connection. The pins and sensor itself ship individually, so in order to use the pins, we first soldered the pins into the corresponding holes on the color sensor. Then, we connected the sensor pins and GPIO ports together with the breadboard

wires described earlier. Figure 15 below shows the five pins on the sensor that are used to connect the sensor to the Raspberry Pi.

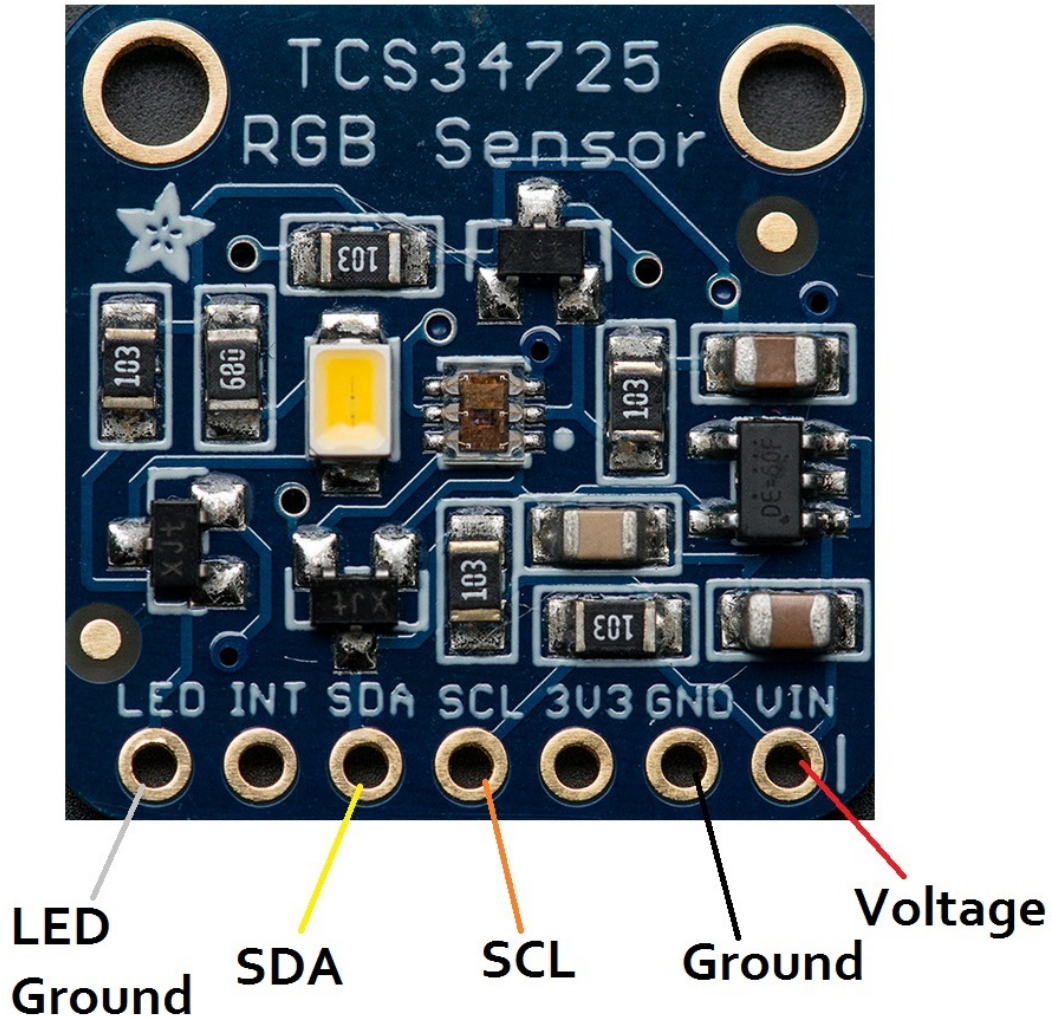


Figure 15: Pins on the TCS34725 color sensor [24]

As seen in Figure 15 above, the TCS 34725 contains pins labeled LED, INT, SDA, SCL, 3V3, GND, and VIN. Each Pin controls a separate component of the sensor and serves a purpose when connected to the Raspberry Pi.

The LED pin is used to control the LED and is connected by a white cable in port 9. When the LED cable is connected from the sensor to the Raspberry Pi ground pin, the LED on the sensor is turned off. When the cable is disconnected from the Raspberry Pi ground, the LED will turn on.

The INT pin was not used for our project. It serves as an external interrupt that will start and stop the sensor's data collection. Because we used the Raspberry Pi and a Python program to control

our data collection, we did not need to use this pin, however it might be more applicable for smaller microcomputers like Arduino.

The SDA and SCL pins are both used to transmit data. The SDA, connected with a yellow wire, and the SCL, connected with an orange wire, are connected to ports 3 and 5 respectively. These pins communicate to the Raspberry Pi the RGBC color values collected by the photoarray on the sensor.

The 3V3 and VIN pins are both designed to power the sensor through a voltage source, however we only used the VIN pin for our project. The VIN pin, connected by a red wire in GPIO port 1, is connected to the Raspberry Pi which is itself connected to a power source (either an outlet or battery source). The wire sends 3.3 Volts from the RPI to power the sensor.

Finally, the GND pin is a ground pin, connected to the RPI by a black wire. The wire connects the sensor to Raspberry Pi's ground in GPIO port 6. Figure 16 below shows the breadboard wires connected to the pins of the TCS34725 sensor.

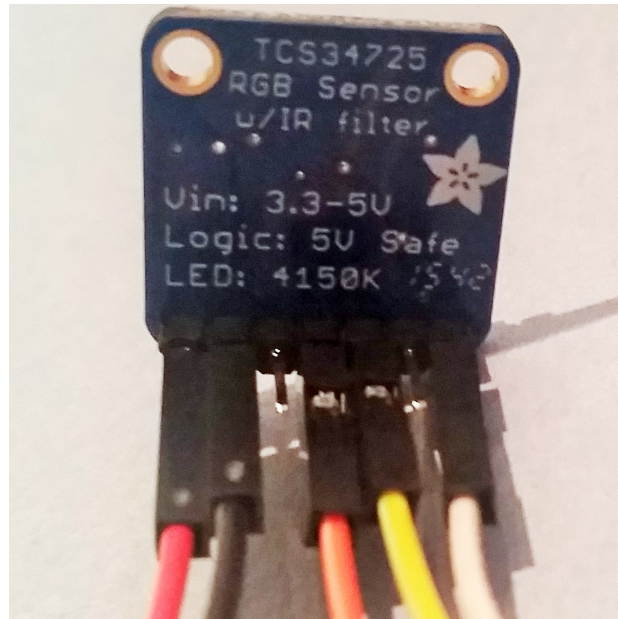


Figure 16: Breadboard wires connected to the TCS34725 color sensor

Once connected to the Raspberry Pi, the TCS34725 requires a prewritten module written in the Python 2 programming language. Because Python 2 is an open source language, we downloaded the module from GitHub, a hosting service for computer programming, and adapted the module for our usage. The module, titled `Adafruit_TCS34725.py`, reads and returns values for r,g,b, and c; the amount of light collected by the red, green, blue, and clear photodiodes respectively. The program also contains 6 prewritten integration times that the program can choose from.

In order for this sensor to work for our project, we had to test its linearity. For example, if the intensity incident is doubled, do the reported values for red, green, blue and clear double as well? We conducted a wide range of tests on the TCS sensor, using neutral density filters to change the intensity of incident light. We tested under several lighting sources and for every color that we expect the TCS sensor to help us identify. An example of this data (taken for a white card under fluorescent light)

is shown in Figure 17. We found that for the most part, the sensor behaved linearly. However, there were some areas where non-linear effects were observed. Some of these effects caused all the values to change by the same non-linear factor. These instances actually don't affect the performance, because as we will discuss later, we are concerned with the ratios between the color values recorded, not the raw values themselves. If all of the values change by the same factor then those ratios are unaffected. An example of this effect is shown in Figure 17, where the last value is lower than expected for a linear trend.

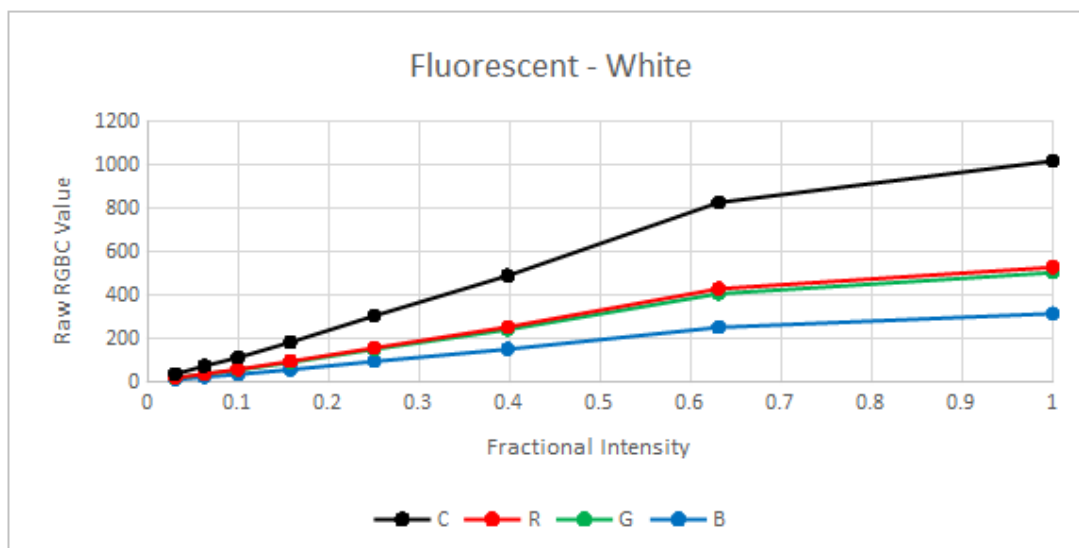


Figure 17: Raw values vs Intensity

We can show conclusively that this kind of effect didn't cause a problem for the final design. By dividing all color values by the clear values, as is done in our Python program (shown in Figure 18), we can look at the data in a new way. In this graph we should expect perfectly flat lines if the sensor was completely linear. We can observe that for the most part, we do see flat lines within a tolerance of about 0.05 for most ratio values. We can see that the non-linear effect at the last data point for Figure 17 is not very apparent, and that dividing by the clear value does account for this non-linear effect.

Some of the data, however, showed some noticeable non-linearity that would change the ratios between these reported raw values. These were mostly under low intensity conditions (seen on the left-most data points in Figure 18), where even the human eye might have trouble detecting colors accurately. Still, we should be aware that lowering intensity values too low could easily result in incorrect color identification because the values coming from the sensor were simply too inaccurate. However, these results were mostly promising, and we were confident that the TCS sensor would be a good choice of color sensor moving forward with the device implementation.

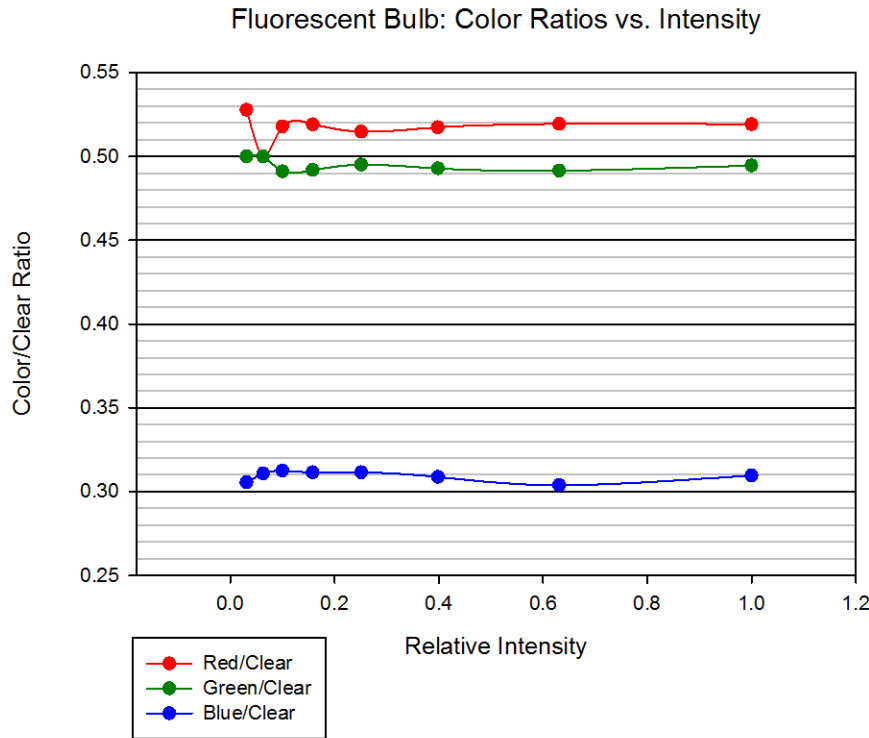


Figure 18: Color ratios vs. intensity for a white card under fluorescent light

## 3.2 Fiber

### 3.2.1 Selecting a Fiber and Numerical Aperture

In order to input light directly onto the TCS34725 color sensor, the team used a plastic optical fiber. This fiber was made by a company that had gone out of business, leaving us unable to find any information on the plastic fiber's specifications. Therefore, the team experimented using laser light collimated through the plastic fiber to make measurements on the numerical aperture, acceptance angle, and core-cladding structure of the fiber. Although information on the plastic fiber was harder to obtain, the team determined the plastic fiber to be the best choice available, because the plastic fiber is sturdy and durable and would not break easily. Additionally, the plastic fiber's diameter was relatively large, which allows the plastic fiber to accept more light than other fibers. This means the TCS34725 sensor will receive a stronger signal, which was desirable for our purposes. After making the decision to move forward with the plastic fiber, we used the photonics lab to explore the properties of this fiber.

The team first determined the numerical aperture of the fiber. To do so, we used a He-Ne laser, the plastic fiber mounted at each end by plastic blocks, and a grid paper. The set up is pictured in Figure 19 below.

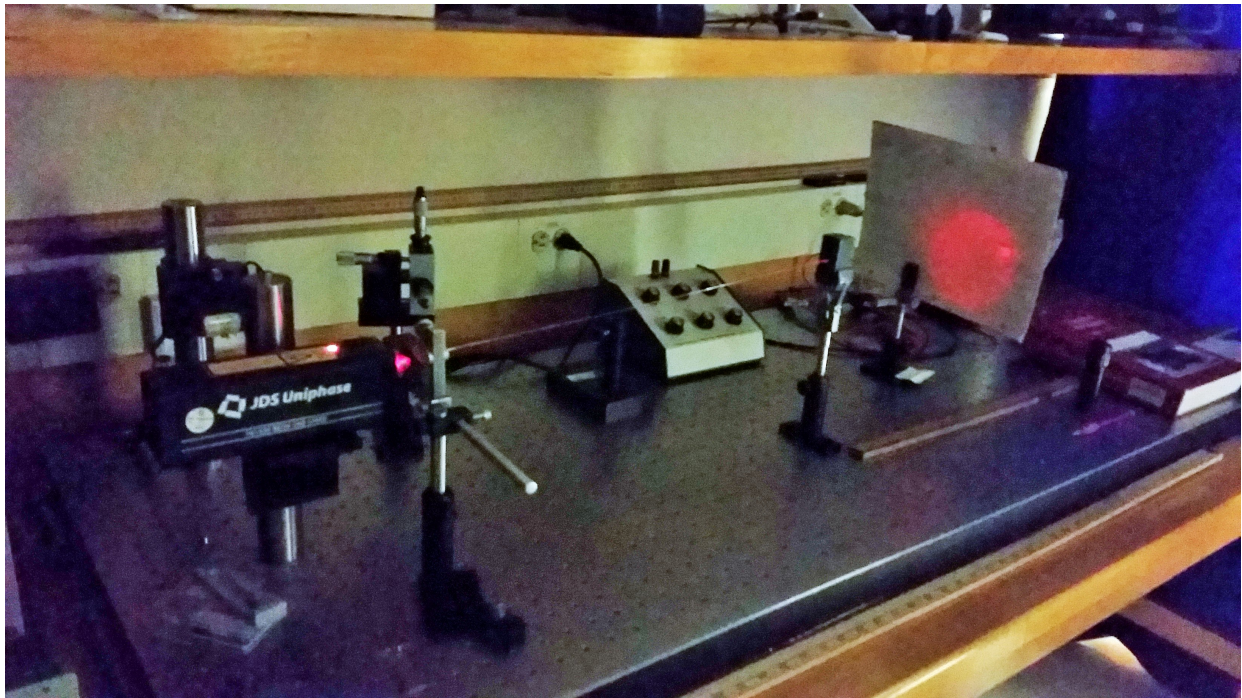


Figure 19: Experimental set up to determine the NA of the plastic fiber

In this experiment, the light from the He-Ne laser was collimated through the fiber and emitted from the fiber onto a grid paper. By measuring the distance between the fiber and the red spot on the grid paper, as well as the radius of the red spot, the team was able to determine the half angle of the fiber. Figure 20 below gives the equation used to find the acceptance angle of the plastic fiber.

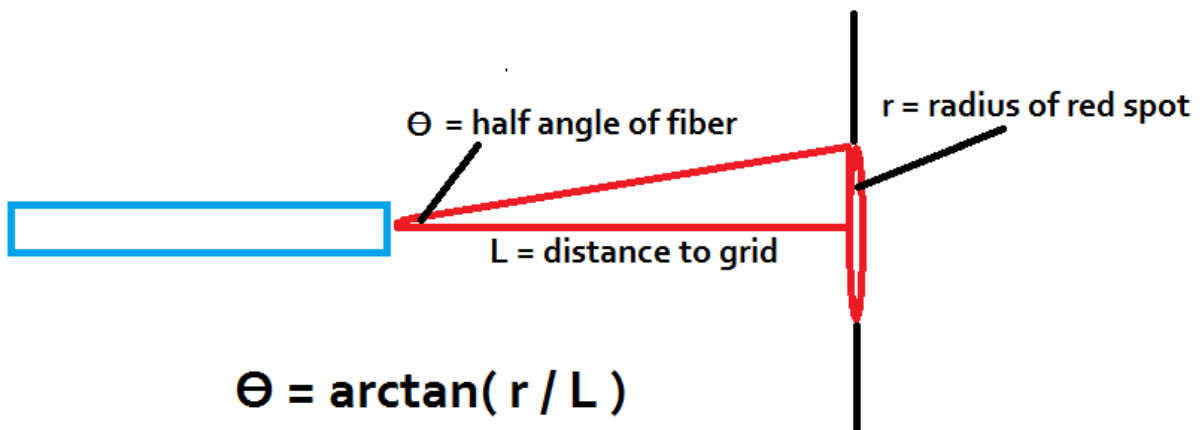


Figure 20: Equation for the half angle of a fiber

The team collected data on the half angle of the plastic fiber for several lengths and found an average half angle of  $13.7^{\circ} \pm 1.5^{\circ}$ . The numerical aperture of the fiber core is given by equation 6 below.



$$NA = n_o \sin(\theta) \quad (6)$$

Here  $n_o$  is the index of refraction of the outside medium. For our experiment, the medium outside the fiber was air which has an index of refraction approximately equal to 1. The average NA found was  $.24 \pm .015$ . We found this NA to be acceptable for the purpose of our project.

### 3.2.2 Core-Cladding Structure of the Fiber

The team found that by examining at the plastic fiber under a microscope, it was unclear what the core-cladding structure of the plastic fiber looked like. In Figure 21 below, the fiber is shown emitting LED light under a microscope. In the figure it appears that the entire fiber is lit with LED light. Therefore it seems possible that the fiber does not have a cladding and only a core, or that the incident LED light propagated through both the fiber core and cladding. Because we could not visibly see any cladding on the plastic fiber, we aimed to determine whether the cladding was small or non-existent through experimentation. We explored the fiber structure by measuring the diameter or the core, as well as the diameter of the fiber as a whole.

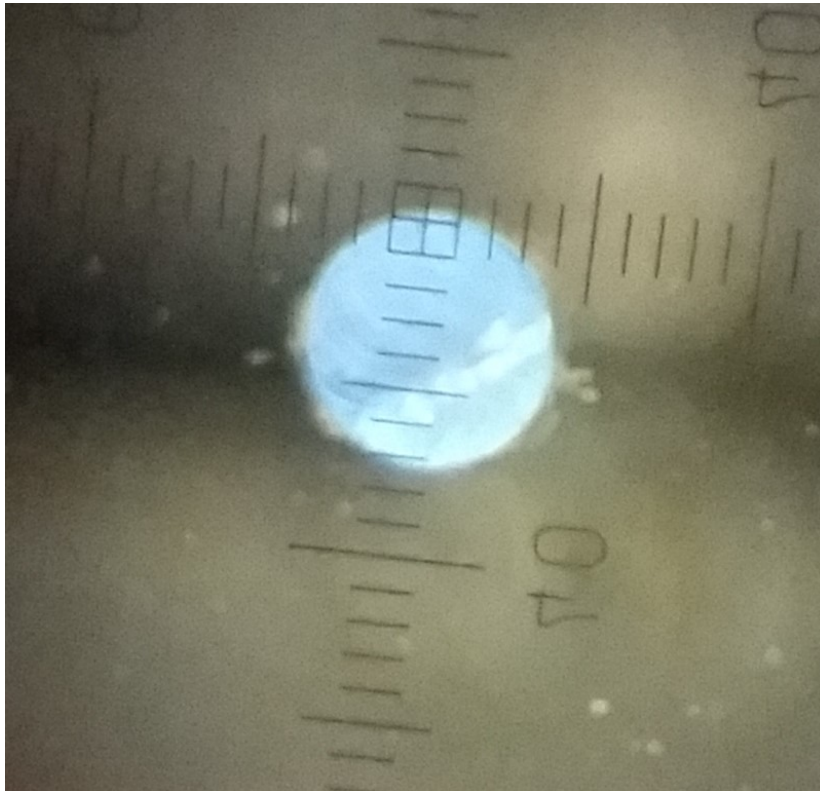


Figure 21: Fiber end lit with LED light

First, we used a reticle and a microscope to measure the diameter of the entire fiber. Figure 22 below shows the reticle placed on top of the plastic fiber.

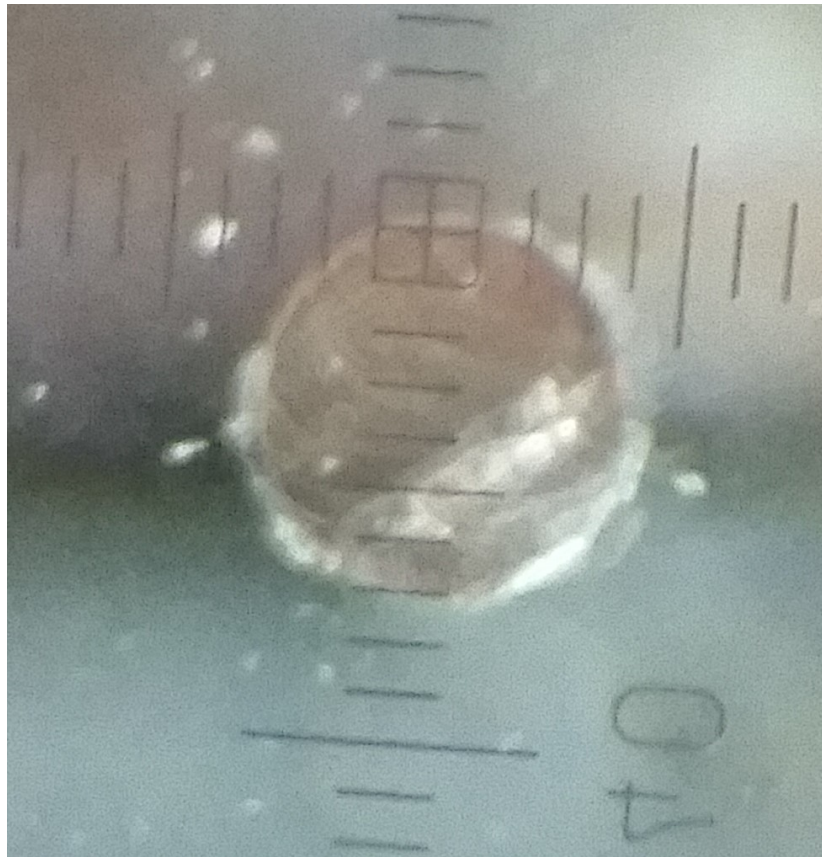


Figure 22: Reticle on top of the plastic fiber

From Figure 22, we estimated that the diameter of the fiber end was about  $700 \mu\text{m} \pm 50 \mu\text{m}$ . Additionally, we measured the diameter of the fiber end with a pair of calipers and obtained a value of  $736 \mu\text{m} \pm 1 \mu\text{m}$ .

Both the reticle and the caliper measurements were only able to determine the diameter of the entire fiber. However, if a cladding existed around the core, the measurements taken so far did not give a value for the core diameter. To find a core diameter measurement, we set up an experiment with the He-Ne laser, the plastic fiber mounted at both ends, a photodiode, a micrometer, and a razor blade.

The fiber was mounted in front of the He-Ne laser and the second fiber end was mounted in front of a photodiode. The photodiode was connected through a resistor to a digital multimeter that read the photodiode's voltage generated by the incident laser light emitted from the second end of the fiber. The micrometer held a razor blade positioned in between the photodiode and the second fiber end as pictured in Figure 23. We used the micrometer to incrementally move the razor blade between the fiber and photodiode until the fiber light was blocked completely by the razor blade. The team noted at what position the photodiode voltage, as read by the multimeter, first began to drop. This position indicated the beginning of the fiber area that emits light. The position at which the voltage read by the photodiode drops to zero indicates the end of the fiber area that emits light.

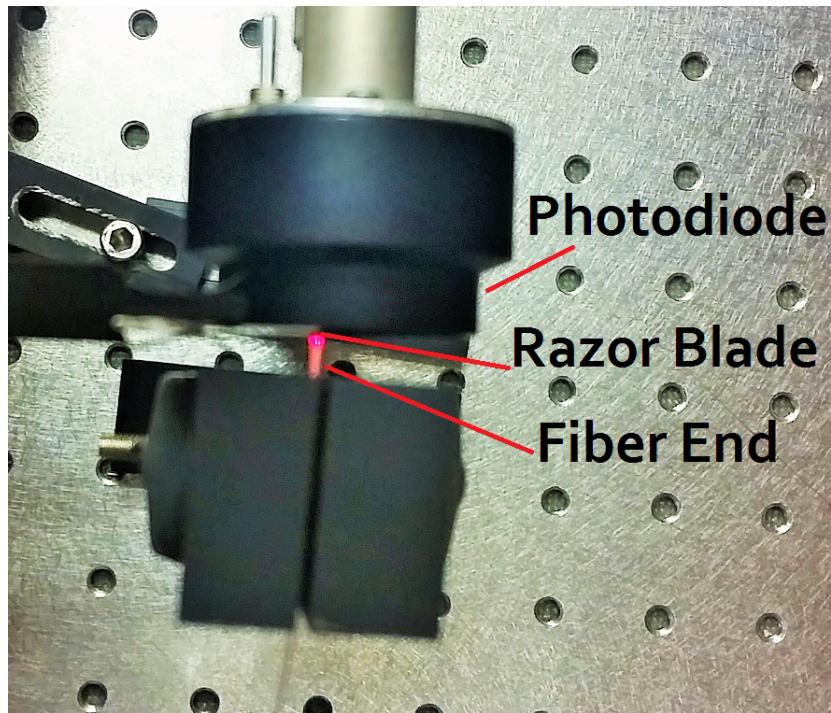


Figure 23: Razor blade positioned between photodiode and fiber end

The difference between the two positions explained above is the diameter of the illuminated fiber region. This region could consist only of the fiber core, or it could be the diameter of the entire fiber if either there is no cladding or if the cladding contains modes that propagate the laser light. We obtained a diameter of  $650\ \mu\text{m}$ , which led us to believe that, because there is about a  $86\ \mu\text{m}$  difference between this measurement and the reticle and caliper measurements, there is thin cladding on the plastic fiber.

Due to the difference in diameter between the illuminated region explained above, and the entire fiber diameter measured with the reticle and calipers, we believed that only a small amount of light, if any at all, propagates through the cladding. However, to further solidify our conclusions, we repeated our experiment for the NA of the fiber. For this experiment, we wrapped the fiber around a post to create small bends in the fiber and then measured the NA of the fiber exactly as before. Figure 24 below shows the set up for this experiment.

If the cladding of the fiber did contain light, bending the fiber would release light from the fiber in the bends. We would expect to see a different numerical aperture if this were the case. From this experiment we found an average NA of  $.23$  whereas the NA we had measured with no bends was  $.24$ . Because the two numerical apertures found are fairly similar, we believe that there is no light propagating in the cladding for any significant distance. Therefore we conclude that the diameter of the fiber core is about  $650\ \mu\text{m}$  and the diameter of the entire fiber (core plus thin cladding) to be about  $736\ \mu\text{m}$ .

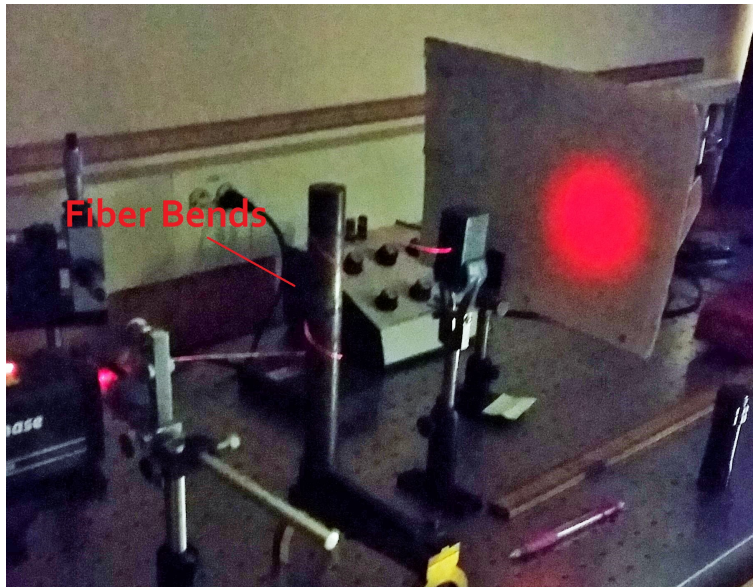


Figure 24: Measuring the NA of the plastic fiber with bends in the fiber

### 3.2.3 Sensor to Fiber Alignment

To approach the problem of aligning the fiber and the sensor, we compared the diameter of the fiber to the size of the photodiode array on the TCS34725 color sensor. We wanted to determine whether we would have to focus light from the fiber onto the photoarray with a lens. As we had mentioned earlier, we determined the diameter of the photodiode to be about  $650\ \mu\text{m}$ , and the photodiode array on the TCS34725 color sensor is  $409 \times 369\ \mu\text{m}$  as listed on its product page.<sup>3</sup> Therefore, a circle of diameter  $548\ \mu\text{m}$  or larger would cover the entire photoarray. The photoarray and fiber are pictured in Figure 25.

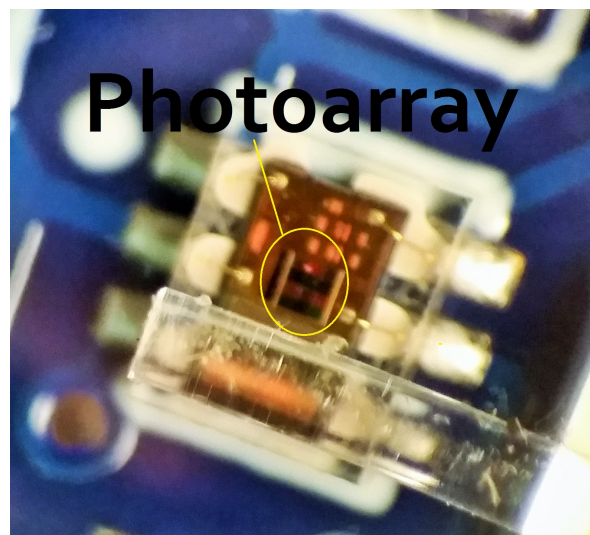


Figure 25: Plastic Fiber and TCS34725 Color Sensor with photoarray

<sup>3</sup>Adafruit product page: <https://www.adafruit.com/products/1334?gclid=CPj8xoXH38kCFUskgQod9g4BNA>

If the fiber diameter was much larger than the photoarray, a percentage of the source light collected by the fiber would miss the photoarray. This would cause a problem because it limits the size of objects that we are able to see and collect light from. Because we found the photodiode array and the fiber to be of comparable sizes, we can simply position the fiber flush against the photoarray. We will use a micrometer to incrementally adjust the x, y, and z position of the fiber and optimize the amount of light transmitted from the fiber onto the photoarray.

We created a three-component system to secure the fiber and correctly align it with the sensor. The first part simply encases the TCS34725 color sensor. Parts two and three are both identical horizontal blocks. Part three, the top block in Figure 26, is attached to part one with screws that run through the TCS34725 color sensor, part one, and part three. Part two has a groove that is lined up with the color sensor's photoarray to secure the fiber in place. Figure 26 shows how parts one, two and three are aligned.

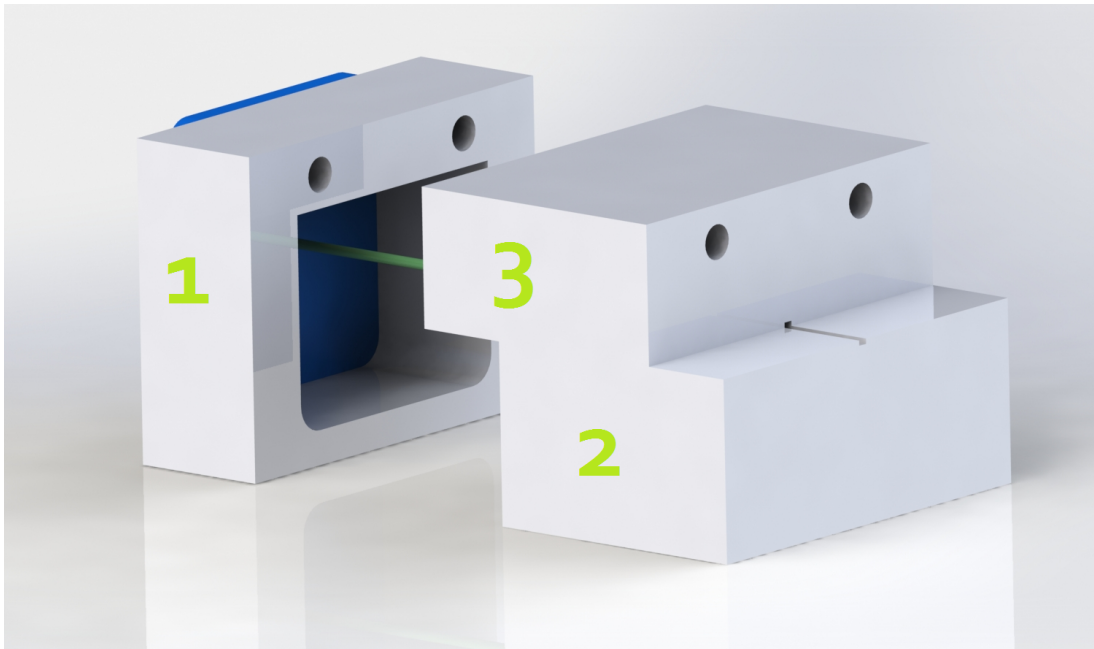


Figure 26: Fiber holder assembly

When parts two and three are wedged together the fiber is secured. To clamp the fiber holder all together, parts two and three are wedged between two identical rectangular blocks of PVC plastic. The bottom block is glued to the bottom of the electric project box which we used to encase our device/footnoteThe complete layout of our device is detailed in Section 3.4, flush against the box's front. We engraved a depression into the bottom block so that part two would securely fit into the block. The depression is not flush against the box front, and only extends the length of part two of the fiber holder. By doing this we allowed the fiber to bend before it reaches the hole in the box front.

The top block of PVC lays on top of the fiber holder and effectively clamps parts two and three together by screws that extend through the top and bottom PVC plastic blocks. We had the option of simply screwing the fiber holder itself together, however, once we do so the alignment becomes

permanent. It is imperative for the fiber to be properly aligned on the color sensor's photoarray so as to collect as much fiber light as possible. Because of this we decided to create the clamping blocks to make the alignment adjustable. Figure 27 shows the clamping of the two PVC plastic blocks with the fiber holder wedged in between.

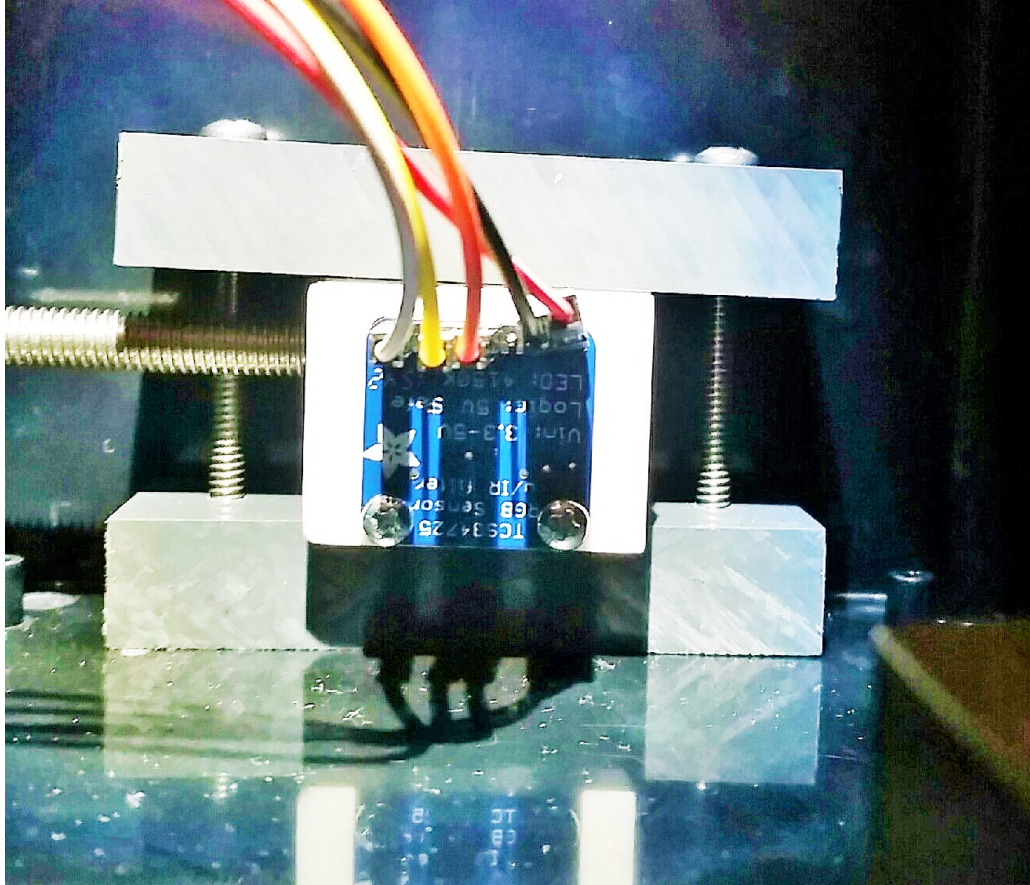


Figure 27: Fiber holder clamps

To align the fiber on the photoarray, we make incremental changes in the fiber's position and observe the resulting changes in recorded 'C' values. To assist in our alignment, we connected a screw through part three of the fiber holder and through the side of the project box. The screw is adjustable so it can be turned to adjust the y position of the fiber. This process is similar to the concept of a linear worm gear and greatly simplifies our alignment process. The z position of the fiber is fixed by clamping the fiber holder together and the x position can be adjusted by hand before clamping the holder.

Finally, the fiber extends from the fiber holder through the box through a small hole drilled in the front of the box. Our fiber length is about three feet (.914 m) to allow the user to easily point the fiber at objects without moving the project box. The fiber hole in the front of the box is where we are most concerned about excess light leaking into the box as any light through the fiber hole is directly incident on the color sensor's photoarray (as the photoarray faces the fiber hole). To eliminate as much light leakage as possible we put putty around the fiber hole. With the putty in place, our

leakage to signal ratio is about 6.8%. Additionally, we tried covering the lights on the Raspberry Pi, the screws holes in the electric project box, as well as the cord slots for the speaker. However, we only noticed a significant change when we covered the fiber itself with black cloth. Figure 28 below shows the percentage of light leakage for various methods of reduction.

Coverage	White Card (C Value)	Black Cloth (C Value)	Leakage Percentage
Putty on Fiber	6177	394	<b>6.38%</b>
+Putty on Cord Slots	5580	414	<b>7.42%</b>
+Screws	6188	446	<b>7.21%</b>
+RPI Lights Covered	6098	372	<b>6.10%</b>
+Half of Fiber Covered	6000	263	<b>4.38%</b>
+All of Fiber Covered	5326	1	<b>.00005%</b>

Figure 28: Leakage percentages for varying degrees of coverage

As the figure shows, the light leakage was significantly reduced when half of the fiber was covered and nearly completely eliminated with the entire fiber covered. This leads us to believe that unwanted light is somehow getting into the fiber along the length of the fiber. Though we do not know exactly why this seems to be happening, we suspect it could be due to light incident on the edges of the fiber cylinder becoming trapped in skewed modes or cladding modes and traveling the length of the fiber. Usually, light entering the fiber from the sides is not significant. However, we have a relatively short length of fiber, and the light we are attempting to capture at the end is very similar in intensity and distribution to the light incident on the fiber sides (as opposed to a laser or diode source). Although the cause is still not certain, the remedy is simple and effective. We address this issue by covering the entire fiber with black heat shrink tubing, and by doing so we eliminate all excess light.

### 3.3 Aperture

We implemented an aperture to limit the acceptance angle of the fiber allowing our device to identify objects that are smaller and farther away than what was possible without an aperture. We had the choice of making the aperture adjustable or fixed. For simplicity, we decided to keep the aperture fixed to a certain diameter.

To create an aperture-like component for our fiber, we decided to create a sheath that encases and extends past the fiber end. The sheath forces the fiber into collecting light from a more focused area. We can adjust the effective acceptance angle by changing the distance between the fiber end and sheath end. We had the freedom to choose these lengths based on what we deemed reasonable for our project. Figure 29 below shows the design of our aperture.

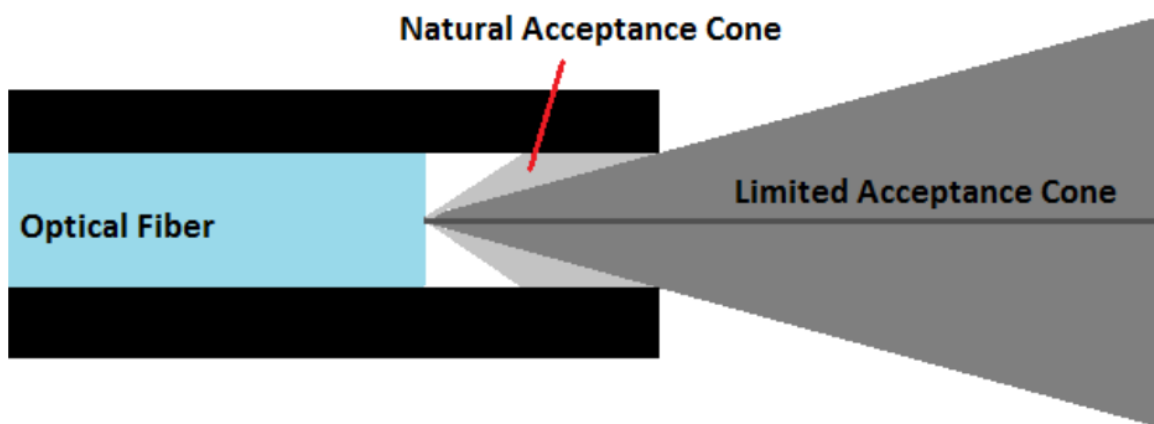


Figure 29: Diagram of sheath-like aperture

The aperture sheath was created using heat shrink tubing, therefore the diameter of the sheaths end is approximately equal to the fiber's diameter which was previously found to be about  $736 \mu\text{m}$ . To determine the optimal distance between the sheath's end and the fiber end, we took into consideration two things; how small can we make the aperture's opening and still achieve a sufficient RGBC signal, and what ratio of object size to object distance is reasonable and desirable for our device.

We decided that an aperture that limits the fiber's acceptance full angle to approximately  $15^\circ$  would be appropriate for our device. Therefore  $d$ , the distance between the fiber end and aperture end is about 3 mm. This acceptance angle allows a user to see a 21 cm object (a standard sheet of paper) 80 cm away.

As stated above, heat shrink tubing was used to make our device's aperture. Figure 30 below shows two pieces of fiber covered by the heat shrink tubing. The tubing is black and matte so the tubing should not cause excess light to reflect into the fiber. The tubing material is fairly durable and malleable, however it is very sensitive to heat so it must be kept out of high temperatures.



Figure 30: Samples of plastic tubing covering our fiber



To create the tubing pieces for our fiber, we first heated the tubing on a an extra piece of fiber. We use an extra piece of fiber to create the sheath because we found that the high temperature from the heat gun ended up melting our fiber. Once the tubing was shrunk down to the fiber diameter, we stripped the tubing from the fiber. The tubing could then be used to encase the fiber we use for our device. We position the tubing sheath so that there is a space between the end of the fiber and the end of the iris (which we defined as  $d$  in equation 5 above). To keep the tubing in place, we will adhere the end of the sheath to the fiber using glue or electrical tape.

## 3.4 Device

### 3.4.1 Device Components

To make our device portable, we composed our device from five main components encased in a plastic electric project box pictured in Figure 31 below. The electric project box, dimensions 8-1/4" length x 5-9/64" width x 3-7/64" height, consists of a rectangular box base with a removable top. We found that the box's dimensions, along with the removable lid served our purposes well as it allows for enough space inside the box to connect and disconnect components as well as allowing room for the wired connections to bend.



Figure 31: Electric Project Box from BUD Industries [3]

The portable components in and on the box include the Raspberry Pi, the TCS34725 color sensor, a battery source, a wireless keypad and a portable speaker. The Raspberry Pi serves as the center of

our device as all other components are connected to the RPI.

The TCS34725 Color Sensor along with our plastic fiber collects light input data and sends this information to the Raspberry Pi through wired connections to the RPI's GPIO pins. The TCS34725 Color Sensor is aligned with the fiber so that light from the fiber is output evenly onto the color sensor's photoarray. Once aligned, the fiber and TCS34725 Color Sensor are clamped using blocks of PVC plastic. .

The user interacts with the Python program by entering numbers on a portable keypad. Our keypad connects to the Raspberry Pi through a wireless USB module. The keypad contains only numbers 0-9, enter, backspace, and other punctuation options.

To output color information as sounds, we use a portable USB powered speaker. The speaker is powered by the Raspberry Pi through a connection in one of the RPI's USB ports, and sound information is sent from the Raspberry Pi to the speaker through the headphone jack. The speaker is mounted on the top of the project box lid so that a user can adjust the speakers volume using its dial.

The project's components are powered by a small power bank inside the box. The power bank is connected only to the Raspberry Pi through a USB port, as all other components are then connected to the Raspberry Pi. The power bank supplies 5 volts to the RPI and is rechargeable. Using the power bank, the device is able to run continuously for about 4.8 hours.

In addition to the device's portable component's, the user also has the option of connecting to a monitor using the Raspberry Pi's HDMI port, as well as using a wired or wireless mouse through a USB port on the RPI.

### **3.4.2 Device Layout**

Our electric project box was large enough that we had plenty of room for our power bank, Raspberry Pi, and TCS34725 color sensor to fit easily in the box along with their wires. Our Raspberry Pi is mounted to the bottom right corner of the project box's removable lid. We mounted the RPI with standoffs to allow for easy access to the RPI's four USB ports, power connection, HDMI port, and GPIO pins.

For now, our power bank is attached to the project box with double sided tape. We had many options when it came to choosing the battery source. Therefore we decided not to permanently secure this component to the box. The power bank is taped in the bottom of the project box and the speaker is taped on top of the box. Figure 32 below shows the location of the power bank, Raspberry and Fiber Holder components in the project box.

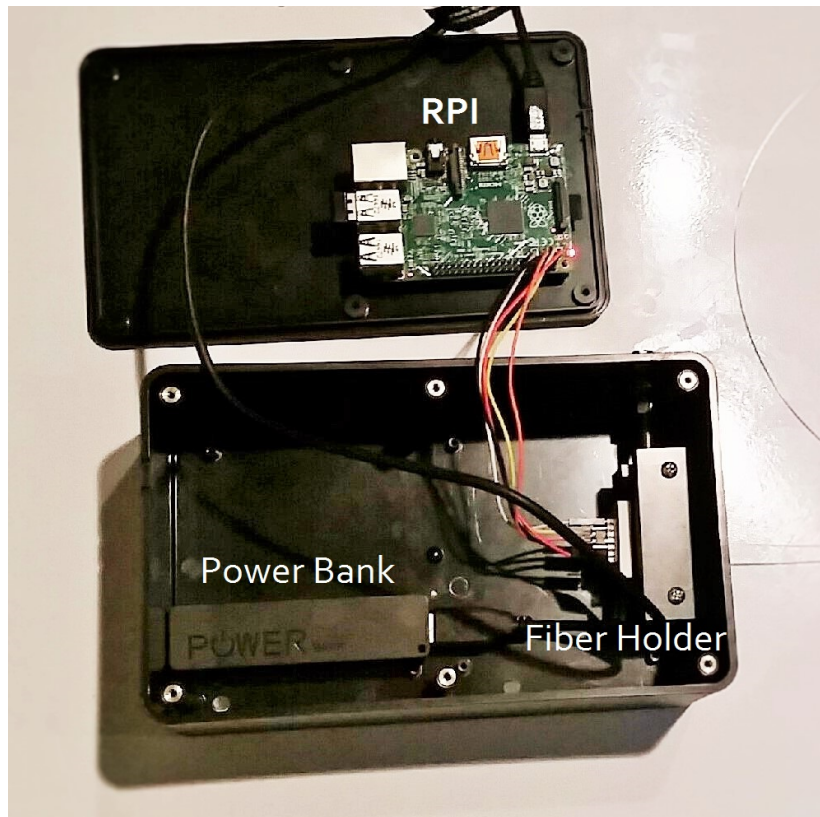


Figure 32: Project box layout top and bottom

The speaker is small enough that it could fit inside the box, but we decided to keep it taped on the box lid instead to allow more room inside the box and to allow the user to better hear the sounds that the speaker emits. The speaker's two wires enter the box through a small slot that was filed directly underneath the lid on the back of the box. Because the cord's slots were filed to the cords measurement and the slots are located behind the TCS34725 color sensor, we do not anticipate a significant amount of light leakage from the slots.

We have set up a HDMI cord connected through the Raspberry Pi's HDMI port as well. The HDMI cord comes out of the box through a slot between the lid and back of the box similar to the slots for the speaker's cords. Though it is not necessary to connect to a monitor to use the device, we do need a monitor to start the Python program. It is also useful to connect to a monitor for demonstration and editing the program's script.

### 3.5 Python Program

The process of analyzing color information and detecting a color takes place in a computer program. We found that Python was the optimal language to use due to its easy and fast implementation and open source module written to access the Raspberry Pi's GPIO ports. We created three modules in Python including a wrapper to perform the task of identifying a color and playing its corresponding tone. The Python program serves four major purposes: to obtain and manipulate RGBC values,

to account for background lighting distributions, to learn colors and backgrounds, and to identify a color.

### 3.5.1 Obtaining and Manipulating RGBC Values

The first step was to obtain red, green, blue and clear intensity values from the TCS34725 sensor and record them in Python variables. To do this, we used an open source module found on GitHub that contained the necessary libraries to communicate with the TCS sensor. Essentially, this module allowed us to call a function to ask for RGBC values from the sensor, and record those values in local variables R, G, B and C.

We wanted to use the raw intensity values for red, green, blue, and clear (unfiltered) light produced by the TCS sensor to create a set of criteria characteristic only to an objects color that can be compared under any background source. So, there are several 'normalizations' that we made to refine the raw data into usable values dependent only on color.

The first consideration is intensity. By dividing the red, green and blue values by the clear value, we created a set of three values that are intensity-independent. Next, we had to normalize for background distribution. To this end, we divided the new values by similar (intensity normalized) values taken from observing a white card under the same background light. Now, our data is normalized for intensity and background distributions. Effectively, we have three values (red, green and blue) that represent how much light was reflected as a fraction of how much was available for reflection. However, from our observations, these values are not necessarily what determines a particular color. What is more characteristic are red-green, red-blue and green-blue ratios. For example, Magenta is characterized by a high red-blue and red-green ratio, and has a green-blue ratio of nearly one. We created these ratios using the aforementioned normalized values, and we are confident that these ratios only change with changing color.

Once the color values have been obtained and normalized, the rest of the program consists of a continuous loop. The loop begins by asking a user for input, prompting with a sound cue. The user should then enter either 4, 5, 6, or 7, which can be entered with a wireless portable USB keypad. The loops consists of a decision tree, carrying out a particular set of instructions based on the user input. If 4 is chosen, a new background is taken. If the user presses 5, the program collects a set of data from the TCS sensor, normalizes it, and determines the color with characteristic ratios closest to it. If the user presses 6 or 7, the program is instructed to 'learn' a color or a background source type, respectively.

Once the selected option has been carried out, the program loops back to the beginning, so a new input can be obtained. The entire process of one action can be executed in 3 to 4 seconds. The speed is most limited by the time needed to make a measurement with the TCS sensor and the time required to play the sound cues, which can't be easily reduced.

### 3.5.2 Account for Background Lighting Distributions

As we mentioned before, the spectral power distribution of light varies significantly for common lighting sources. "Taking a background" refers to the act of accounting for these changes. To take a background, the user first points the fiber at a black cloth, and then at a white card. The black cloth value represents any excess light reaching the sensor. We eliminate this effect by subtracting the black cloth "leakage" values from the white card values. The white card values are representative of the spectral power distribution of the current lighting environment.

Overall, this set of data represents the background distribution. The program matches this distribution to the closest source type (Tungsten, CFL, LED, Fluorescent or Sunlight)<sup>4</sup>, and uses these values for normalization as discussed in the previous section. A sound cue consisting of two notes is also played at this time, with a unique sequence corresponding to the source type.

### 3.5.3 Learn Colors and Backgrounds

In order to identify colors, we decided that we should compare unknown values to "learned" values that the user had previously defined. Therefore, we needed a way to tell the program the RGB values it should expect for the detectable colors. We implemented a basic machine learning technique to allow the device to "learn" colors.

From our research in color theory we hypothesized that background distributions and intensity could be completely normalized, and therefore it would be possible to record color values under one background source and use them to determine colors under another source. Although this theory seems sound, we found in practice that large changes in background lighting still had a noticeable affect on these ratios. However, small changes in background could still be effectively accounted for with normalization. To fix this, we created a different set of criteria for each type of background source; this meant that we had to learn background source types as well. Slight changes between instances of the same source are still accounted for with normalization, but large changes completely change the set of values used for comparisons.

The implementation for this part of the code was done in a separate module titled LearnColorRatios. This module contains function definitions which can be used by the main program in order to assist in completing repetitive tasks. The LearnColorRatios module revolves around receiving normalized color ratio information as function arguments, and either saving that information in csv files or reading existing information from csv files and using it to determine colors or background source types.

As mentioned before, we have created separate libraries for color ratio storage for each type of background source. When the user presses 6 to learn a new color, they must also tell the program what color is being learned by entering a key (such as 'M' for Magenta). The normalized ratios are appended to the csv file for the corresponding color in the current source type library. The program then automatically updates a second file, named 'colorinfo.csv', which contains the mean values of

---

<sup>4</sup>The comparison will be discussed in Section 3.5.4

these ratios for each color. A separate 'colorinfo.csv' exists for each source type, and the program uses the information from this file when determining a color. A similar system exists for learning background source types, accessed when the user enters 7. This implementation is very flexible and dynamic, allowing the user to 'teach' several instances of a color to improve the devices accuracy. The process of learning a color is shown in Figure 33.

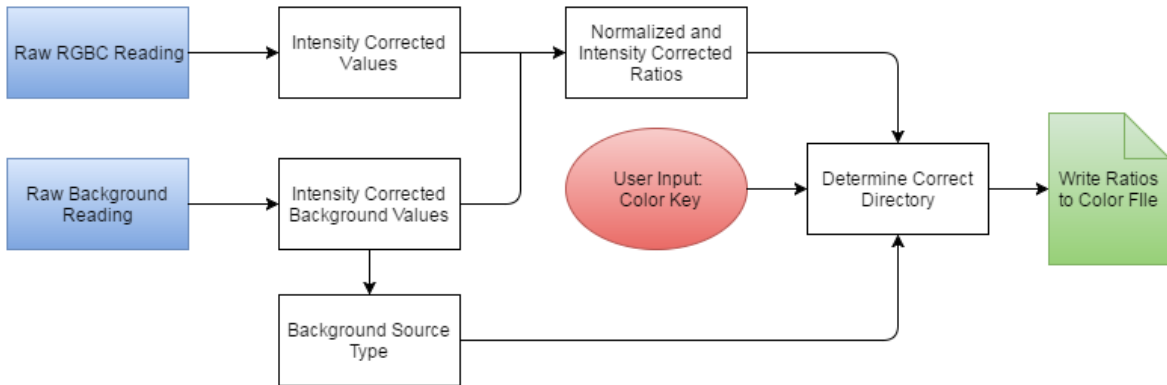


Figure 33: Flowchart of the process of learning a color

### 3.5.4 Identify a Color

If the user presses 5, the main program attempts to determine a color. The LearnColorRatios module contains a function, compareColor, which handles the decision. This function requires three normalized values (red, green and blue) as input. It also takes the current background source type. With this information, it opens the corresponding instance of 'colorinfo.csv' and reads the information from it. To decide which color has the closest characteristic ratios, the function loops through each possible color and computes the variable called Lms. Lms is calculated by raising the difference between the detected object's color ratios and the known characteristic color ratios to the fourth power, and then summing those values. It is similar to calculating least mean squares, except we use the fourth power to bias towards cases where each ratio is only slightly off, and against cases where any one ratio has a moderate to large difference. Our research has shown that even just one ratio being significantly off is indicative that the color in question is not correct. A similar process is used to identify background source types. This process is illustrated in Figure 34.

The program keeps track of the lowest Lms, and correspondingly, the best color. Here, we implemented a clause that overrides a choice of white if a second best color had an Lms score that was within 30 percent of the Lms score for white. This reduces false white identifications and has had no negative effect on correct white identifications. We have noted that if the color in question is white, the Lms score will often be extremely small, as white has characteristic ratios of unity when normalized to the background. Another way of thinking about background normalization is that we are *defining what white is* as a benchmark for other colors to be measured against. Therefore, we

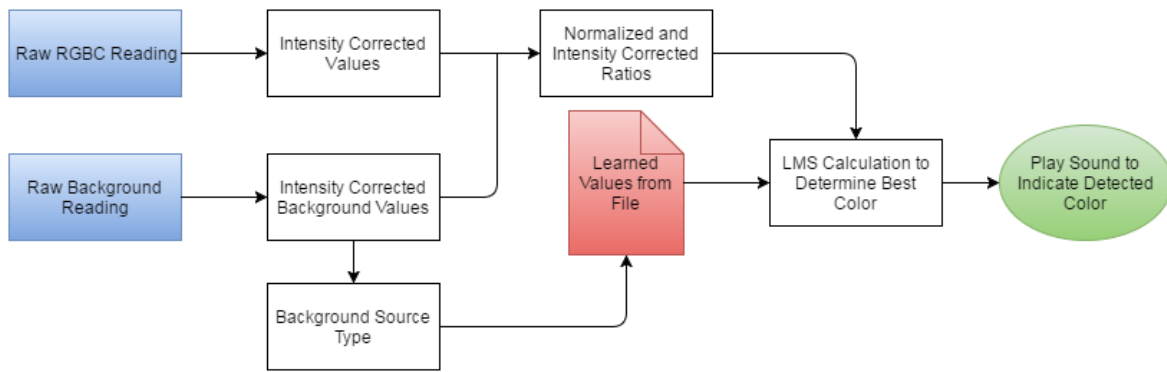


Figure 34: Flowchart of the process of detecting a color

can be very strict about the requirements for detecting white, as the characteristic values should be identical to those of the background.

### 3.6 Audio Output

This portion of the project has a simple objective. The user should be able to rely on sound cues alone to operate the instrument and to determine which color had been detected. We also decided it would be important for the user to know the type of background source from a different set of sound cues. These sounds should be distinct to the human ear and require minimal ear training to remember and recognize, and the user should be able to tell immediately whether the device was ready for operation, had just detected a color, or was awaiting an input command.

Since we used Python to detect colors, we decided to continue using the program to handle playing musical tones. This module uses the pygame library to play .wav files based on the color detected. For each color, a designated wav file exists to denote that particular color. The sounds played when a color is detected are composed of a three note sequence. The first note is always middle C, and the second note is always higher than the first. This is unique among our sound cues, letting the user know right away that a color is being described. The three note sequences all use basic melodic structures that are easily distinguished; for example, the sequence for red is C, D, E, the same as the beginning of the nursery rhyme Frere Jacques. Every sequence is equally as simple melodically, allowing even someone with no formal musical training to use the device with an easy learning curve. A table of the sequences and the colors they represent is shown below. The 'Melodic Descriptions' are just examples in well known songs where these three note melodies can be found, and there are likely many more.

<b>Three-note Sequences by Color</b>		
<b>Color</b>	<b>Sequence</b>	<b>Melodic Description</b>
Magenta	C-A-B	"The Way You Look Tonight" - "Just Thinking"
Red	C-D-E	Frere Jacques
Orange	C-D-F	"Valerie" opening notes - "Well Sometimes"
Yellow	C-E-G	C major chord arpeggio
Green	C-A-G	Downward Progression - used in ...
Cyan	C-F-E	... "You're Never Fully Dressed ...
Blue	C-D-C	... Without a Smile"
Purple	C-F-C	Jeopardy theme beginning
Brown	C-B-Bb	Chromatic scale (descending)
White	C-G-C	Octave above middle c

A similar system exists for the detection of background source types. Here, every sequence is only two notes long, and each one starts an octave above middle C. The second note is always lower, creating an interval that is easy to hear and distinguish. We also created a sequence G, C that is played when the device is ready for input, and a single tone of G is played when background is being obtained. These unique cues help the user know when and how to interact with the device. The table below shows the note sequences for each background source.

<b>Other Sound Cues</b>		
<b>Purpose</b>	<b>Sequence</b>	<b>Description</b>
Fluorescent Source	C-F	Perfect fifth interval
CFL Source	C-E	Minor sixth interval
LED Source	C-D	Dominant seventh interval
Tungsten Source	C-G	Perfect fourth interval
Sunlight Source	C-A	Minor third interval
Ready for Input	G-C	User should input command
Background Read	G	Single note when reading values
Startup	E-C-G	Quick, only plays once

All of these sequences were made using the free program Audacity, a simple audio editing suite. The tones are simple sine waves generated at a frequency corresponding to a note in the 12 tone chromatic scale in western music. Each tone lasts for half a second to give the user enough time to hear and recognize it. With this implementation we were able to realize the goal of communicating device status and color information to the user by taking advantage of the ear's natural strength in melodic recognition.



## 4 Results and Conclusions

### 4.1 Accuracy

We can test the accuracy of our device by attempting to identify colors at increasing distances. For this experiment, we keep our aperture length, bulb intensity, and card size fixed and increment the distance between the fiber and the card. As we mentioned earlier, we chose to use an aperture that limits the fiber's acceptance angle to about  $15^\circ$ . Therefore, we should be able to exclusively see a card with a 21.5 cm diameter up to a distance of about 80 cm. Past 80 cm, the fiber begins to see not only the card, but also the surface or background surrounding the card. Therefore, we say that a perfectly functional color sensor should begin to fail at 80 cm.

We repeated this experiment for all background lighting sources that we currently have learned: Sunlight, Tungsten Bulbs, CFLs, Fluorescent Lamps, and LEDs. The fiber to card distances were 7 cm, 20 cm, 35 cm, and 50 cm. The detected objects were the same color cards that were used to learn the colors.

Overall, we had generally satisfactory results. We were able to correctly identify colors under all light sources with a fiber to card distance of at least 35 cm. Only at a distance of 50 cm did we start failing to correctly identify colors. Under the CFL, Fluorescent and Tungsten lights, the device was unable to identify red at a fiber to card distance of 50 cm as can be seen in Figure 35. Although, the device was still able to correctly identify red under sunlight and LED light, the LMS calculation for these identifications was relatively high (on the order of a tenth) which means that the device was close to failing and mistaking the red card for orange or magenta. We consider the fiber distance to be the main factor causing our device to fail as all other conditions were kept constant.

Bulb	Pass/Fail	Identified Color
Fluorescent	F	Orange
LED	P	-
CFL	F	Orange
Tungsten	F	Magenta
Sunlight	P	-

Figure 35: Ability to identify red card at a distance of 50 cm

All other colors, however, were correctly identified even at 50 cm. It is worth noting that the incorrectly identified colors (orange and magenta) can be considered shades of red. This suggests that these failures were not due to some kind of signal to noise problem. The sensor was still generally

able to detect the shade, but the ratios were significantly different than the correct values. The best explanation calls on our earlier results when testing the sensor for linearity. The aforementioned non-linear problem areas could very well explain these results, although we do not have a conclusive answer for the root cause of these color identification failures.

## 4.2 Limitations

Although we can be generally satisfied that our device can accurately identify all colors that it has been taught, there are still limitations to the device. The accuracy results illustrate difficulty identifying objects that are farther away. At a distance of 7 cm, the LMS calculation for the correct color was very low (on the order of a millionth). As we increase the distance between the card and the fiber, the LMS numbers steadily grew, displaying that the device was having a harder time detecting the object's color. We can say that the main limitation to this device is the distance from the object, and that we experience color identification failure at a distance closer than theoretically predicted. While there are inherent distance limitations to the design, we were not able to optimize accuracy to these limitations, and we experienced identification failure at significantly closer distances.

The other main limitation for this device in terms of accuracy is the shininess of the examined object. Shiny objects reflect light very differently than their matte counterparts; the distribution observed contains much more white light, which makes color identification harder. We experience this as glare in our everyday lives. Our eyes use their spatial resolution to look around glare and still detect object color, but this device cannot resolve spatially at all. A different implementation, perhaps using a digital camera, would allow the developers to have the tools to tackle this issue. However, it is an inherent limitation for our design.

Finally, this device does not have the necessary ease of use to be produced as an actual product for typical users. The color detecting program has to be started manually, and is not a stand alone executable. It needs the Python IDLE environment to run. Furthermore, a user needs to use a mouse and a display to start the program. We expect that these limitations could be addressed without too much difficulty, but the current group felt that their lack of programming knowledge and experience prohibited them from creating a useful and elegant solution.

## 4.3 Conclusions and Future Work

In conclusion, we have successfully met the objectives that were set at the beginning of this project. However, our project can be significantly improved by addressing the device's limitations as well as expanding the device to accomplish more.

First the issue of distance should be addressed. During the implementation of this project, we considered using a lens in order to be able to focus on small objects that are far away. However, for the device to be able to see objects that are both near and far, the lens should be retractable; effectively imitating the focusing power of the eye. An elegant solution could consist of a lens that

automatically adjusts based on the distance of the object being examined.

Additionally, other facets of the device design can be improved. The aperture that we used was fixed to an acceptance angle of about  $15^\circ$ , but we found that the sacrifice of light intensity from the implementation of an aperture would not affect color detection until a limited acceptance angle of about  $4^\circ$ . Therefore, an adjustable aperture could be implemented to expand the distance of objects that the device can naturally detect, even without the help of a lens. Secondly, the device can be made even more compact and portable by using a smaller project box as well as using the newest model of the Raspberry Pi (Model Zero) which is significantly smaller than the Model 2B that we used.<sup>5</sup>

While this design produced a reasonably successful implementation, there are undoubtedly many other ways to resolve the problem of color detection. We hope that others will continue to try new and creative ways to approach this project and reach new solutions.

---

<sup>5</sup>The Raspberry Pi Zero (65mm x 30mm x 5mm) was released recently with much smaller dimensions than our model, the Raspberry Pi 2 B (85.60mm x 56mm x 21mm).

# 5 Appendices

## 5.1 Python Code

### Main Program Module: ListeningtoLight\_Final.py

```
# =====  
# Main Program for Listening to Light MQP  
# Authors: Ryan Lang and Amanda Varrichione  
# =====  
  
# This program accesses the Adafruit TCS34725 sensor from Raspberry Pi GPIO ports.  
from Adafruit_TCS34725_Exp import TCS34725  
from time import sleep  
import math  
  
import ColorSounds_Final as sort  
import LearnColorRatios_Final as LC  
  
# Initialize the TCS34725 and use default gain  
tcs = TCS34725(integrationTime=0x00, gain=0x03)  
  
tcs.setInterrupt(False) # False sets sensor to active state  
  
# intTimes defines possible integration times corresponding to settings in  
# Adafruit_TCS34725_Exp  
intTimes = [0xFF, 0xFC, 0xF8, 0xF4, 0xF0, 0xEC, 0xE8, 0xE4, 0xE0, 0xDC, 0xD8, 0xD4, 0  
xD0, 0xCC, 0xC8, 0xC4, 0xC0, 0xBC, 0xB8, 0xB4, 0xB0, 0xAC, 0xA8, 0xA4, 0xA0, 0x9C, 0x98  
, 0x94, 0x90, 0x8C, 0x88, 0x84, 0x80, 0x7C, 0x78, 0x74, 0x70, 0x6C, 0x68, 0x64, 0x60, 0x5C, 0x58, 0  
x54, 0x50, 0x4C, 0x48, 0x44, 0x40, 0x3C, 0x38, 0x34, 0x30, 0x2C, 0x28, 0x24, 0x20, 0x1C, 0x18, 0  
x14, 0x10, 0x0C, 0x08, 0x04, 0x00]  
  
# backIntTimes sets integration times for each possible background  
backIntTimes = {'Tungsten': 20, 'Fluorescent': 20, 'LED': 20, 'Sunlight': 20, 'CFL':  
20}  
  
tcs.setIntegrationTime(intTimes[20]) # Initializes integration time before obtaining  
background  
  
# Initializes background and light correction variables  
haveBackground = False  
background = ''  
corr = [0.0, 0.0, 0.0, 0.0]  
  
# Lets user know device is ready  
sort.startupSound()  
sleep(1)
```

```

while True:

    # Prompts user for input
    sort.readySound()
    choice = str(raw_input('4_for_background, 5_for_color, 6_to_learn_color, 7_to_
        learn_background:'))
    tcs.setInterrupt(False)

    # Collects data and corrects for light leakage and intensity
    rgb = tcs.getRawData()
    c = float(rgb['c']) - corr[0]
    if c == 0: # Avoids division by zero
        c = 1
    rc = (float(rgb['r']) - corr[1])/c
    gc = (float(rgb['g']) - corr[2])/c
    bc = (float(rgb['b']) - corr[3])/c

    if rc != 0 and gc != 0 and bc != 0: # Don't try in pitch black

        # User asked to set new background
        if choice == '4': # User needs to cover fiber with black cloth
            rgb = tcs.getRawData() # Gets leakage levels
            corr[0] = float(rgb['c'])
            corr[1] = float(rgb['r'])
            corr[2] = float(rgb['g'])
            corr[3] = float(rgb['b'])
            sort.backgroundSound() # Indicate leakage levels obtained
            sleep(1)

        # User needs to point fiber at white card
        raw_input('Aim fiber at white card and press enter to continue')
        print 'Reading background'

        rgb = tcs.getRawData()

        c = float(rgb['c']) - corr[0]
        if c == 0:
            c = 1
        rr = (float(rgb['r']) - corr[1])/c
        gr = (float(rgb['g']) - corr[2])/c
        br = (float(rgb['b']) - corr[3])/c

        print 'Background obtained'
        sort.backgroundSound()

    # Determines background source type

```

```

background = str(LC.findBack(rr , gr , br))
print(background)

haveBackground = True

# Sets integration time to new level based on source type
tcs.setIntegrationTime(intTimes[backIntTimes[background]])
sleep(1)

# Determines an objects color
elif choice == '5' and haveBackground == True:
    print('Finding_Color... ')

    # Normalizes object color values to background values
    Nr = rc/rr
    Ng = gc/gr
    Nb = bc/br
    # Uses normalized values to determine color
    color = LC.compareColor(Nr, Ng, Nb, background)
    print (color)
    sort.Color(color) # Plays sound corresponding to color
    sleep(1)

# Learns a color
elif choice == '6' and haveBackground == True:
    Nr = rc/rr
    Ng = gc/gr
    Nb = bc/br

    # Writes color values to corresponding file
    LC.LearnColor(Nr,Ng,Nb, background)
    print('Learning_Color... ')

    # Updates file containing averages of all colors
    LC.Mean(background)
    print('Color_learned')
    sort.backgroundSound() # Lets user know color has been recorded
    sleep(1)

# Learns a background source
elif choice == '7':
    LC.LearnBack(rc ,gc ,bc) # Writes background values to corresponding file
    print('Learning_Background... ')
    # Updates file containing averages of all backgrounds
    LC.MeanBack()

    print('Background_learned')

```

```
    sort.backgroundSound() # Lets user know background has been recorded
    sleep(1)

# Ends the program
elif choice == '999':
    break

tcs.setInterrupt(True)
sleep(1)

tcs.disable()
```

## Color Learning Module: LearnColorRatios\_Final.py

```
# =====  
# Color Comparison Module for Listening to Light MQP  
# Authors: Ryan Lang and Amanda Varrichione  
# =====  
  
import math  
import csv  
  
# Defines file paths for colors and background sources  
filenames = {'M': 'magenta.csv', 'R': 'red.csv', 'Y': 'yellow.csv', 'G': 'green.csv',  
             'C': 'cyan.csv', 'B': 'blue.csv', 'W': 'white.csv'}  
backnames = {'T': 'Tungsten.csv', 'F': 'Fluorescent.csv', 'L': 'LED.csv', 'C': 'Compactfluorescent.csv', 'S': 'Sunlight.csv'}  
  
# Learns a color by writing r,g,b ratios to the corresponding file  
def LearnColor(Nr,Ng,Nb, background):  
    rgRatio = Nr/Ng # Red to Green  
    rbRatio = Nr/Nb # Red to Blue  
    gbRatio = Ng/Nb # Green to Blue  
  
    # User tells program the object color  
    Color = str(raw_input("What color is this? Enter M,R,Y,G,C,B, or W: "))  
    data = [] # Will receive old file data  
  
    # Opens and records existing file data  
    with open('/home/pi/MQP/'+str(background)+'/'+filenames[Color]) as csvfileread:  
        reader = csv.DictReader(csvfileread)  
        for row in reader: # Saves each row of data in 'data' list  
            data.append({'rgRatio': row['rgRatio'], 'rbRatio': row['rbRatio'], 'gbRatio': row['gbRatio']})  
  
    # Re-writes file with new data added  
    with open('/home/pi/MQP/'+str(background)+'/'+filenames[Color], 'w') as csvfilewrite:  
        fieldnames = ['rgRatio', 'rbRatio', 'gbRatio'] # Header information  
        writer = csv.DictWriter(csvfilewrite, fieldnames=fieldnames)  
        writer.writeheader() # Writes header to first line  
        for line in data: # Writes existing data from 'data' list  
            writer.writerow(line)  
        # Writes new data entry at end of file  
        writer.writerow({'rgRatio': rgRatio, 'rbRatio': rbRatio, 'gbRatio': gbRatio})  
    return  
  
# Identical to LearnColor, but records a background value instead  
def LearnBack(Nr,Ng,Nb):
```



```

rgRatio = Nr/Ng
rbRatio = Nr/Nb
gbRatio = Ng/Nb

Color = str(raw_input("What background is this? Enter T, F, C, S, or L: "))
data = []

with open('/home/pi/MQP/Background/'+backnames[Color]) as csvfileread: # Save
    existing data
    reader = csv.DictReader(csvfileread)
    for row in reader:
        data.append({'rgRatio': row['rgRatio'], 'rbRatio': row['rbRatio'], '
                    gbRatio': row['gbRatio']})

with open('/home/pi/MQP/Background/'+backnames[Color], 'w') as csvfilewrite:
    fieldnames = ['rgRatio', 'rbRatio', 'gbRatio']
    writer = csv.DictWriter(csvfilewrite, fieldnames=fieldnames)
    writer.writeheader()
    for line in data:
        writer.writerow(line)
    writer.writerow({'rgRatio': rgRatio, 'rbRatio': rbRatio, 'gbRatio': gbRatio})
return

# Computes mean for all color values for one background source and records in '
colorinfo.csv' file
def Mean(background):
    data = {} # Initialize data dictionary
    for key in filenames: # Loops for each color file name
        # Initialize mean values and counter
        rgSum = 0.0
        rbSum = 0.0
        gbSum = 0.0
        rgMean = 0.0
        rbMean = 0.0
        gbMean = 0.0
        counter = 0

    # Opens current color file
    with open('/home/pi/MQP/'+str(background)+'/'+filenames[key]) as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader: # Sums rg,rg,gb ratios
            rgSum += float(row['rgRatio'])
            rbSum += float(row['rbRatio'])
            gbSum += float(row['gbRatio'])
            counter +=1
        if (counter) != 0: # Divides by number of ratios in sum
            rgMean = rgSum/float(counter)

```

```

        rbMean = rbSum/float(counter)
        gbMean = gbSum/float(counter)

    data[key] = [rgMean,rbMean,gbMean] # Adds means to data dictionary

# Writes new data file with new information
with open('/home/pi/MQP/'+str(background)+'/'+'colorinfo.csv', 'w') as csvfile:
    fieldnames = ['Color', 'rgMean', 'rbMean', 'gbMean']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    for key in data:
        writer.writerow({'Color': key, 'rgMean': data[key][0], 'rbMean': data[key][1],
            'gbMean': data[key][2]})

return

# Identical to Mean, but computes mean for background values
def MeanBack():
    data = {}
    for key in backnames:
        rgSum = 0.0
        rbSum = 0.0
        gbSum = 0.0
        rgMean = 0.0
        rbMean = 0.0
        gbMean = 0.0
        counter = 0

    with open('/home/pi/MQP/Background/'+backnames[key]) as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            rgSum += float(row['rgRatio'])
            rbSum += float(row['rbRatio'])
            gbSum += float(row['gbRatio'])
            counter +=1
        if (counter) != 0:
            rgMean = rgSum/float(counter)
            rbMean = rbSum/float(counter)
            gbMean = gbSum/float(counter)

    data[key] = [rgMean,rbMean,gbMean]

with open('/home/pi/MQP/Background/colorinfo.csv', 'w') as csvfile:
    fieldnames = ['Color', 'rgMean', 'rbMean', 'gbMean']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    for key in data:
        writer.writerow({'Color': key, 'rgMean': data[key][0], 'rbMean': data[key][1],

```

```

        'gbMean': data[key][2])
    return

# Determines color by comparing current rg, rb, gb ratios to file information
def compareColor(Nr, Ng, Nb, background):
    rgRatio = Nr/Ng
    rbRatio = Nr/Nb
    gbRatio = Ng/Nb

    # Lms represents how close the current colors ratios are to each color file
    Lms = 10000000 # Initialize with impossibly high value
    Lms2 = 10000000 # 2nd best lms
    color = ''
    color2 = ''

    # Opens 'colorinfo.csv' for current background
    with open('/home/pi/MQP/'+str(background)+'colorinfo.csv') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader: # For each color, compare current ratios to known values
            newLms = (float(row['rgMean']) - rgRatio)**4 + (float(row['rbMean']) -
                rbRatio)**4 + (float(row['gbMean']) - gbRatio)**4

            if newLms < Lms: # If current color score is better than the previous best
                color2 = color
                color = row['Color'] # Sets color to current color
                Lms2 = Lms
                Lms = newLms
            elif newLms < Lms2:
                color2 = row['Color']
                Lms2 = newLms

    print(color, Lms, color2, Lms2)

    if color == 'W': # Correct for false white detection
        if abs(Lms - Lms2)/Lms < 0.3:
            color = color2

    return color

# Identical to compareColor, but compares background values to determine correct
background source
def findBack(Nr, Ng, Nb):
    rgRatio = Nr/Ng
    rbRatio = Nr/Nb
    gbRatio = Ng/Nb

    Lms = 10000000

```

```

back = ''

with open('/home/pi/MQP/Background/colorinfo.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        newLms = (float(row['rgMean']) - rgRatio)**4 + (float(row['rbMean']) -
            rbRatio)**4 + (float(row['gbMean']) - gbRatio)**4

        if newLms < Lms:
            back = row['Color']
            Lms = newLms

if back == 'T':
    back = 'Tungsten'
if back == 'F':
    back = 'Fluorescent'
if back == 'L':
    back = 'LED'
if back == 'C':
    back = 'CFL'
if back == 'S':
    back = 'Sunlight'

return back

# Clears and formats color files
def initializeColorFiles(background):

    for key in filenames:
        filename = filenames[key]
        with open('/home/pi/MQP/'+str(background)+'/'+filename, 'w') as csvfile:
            fieldnames = ['rgRatio', 'rbRatio', 'gbRatio']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()

    return

# Clears and formats background files
def initializeBackFiles():

    for key in backnames:
        filename = backnames[key]
        with open('/home/pi/MQP/Background/'+filename, 'w') as csvfile:
            fieldnames = ['rgRatio', 'rbRatio', 'gbRatio']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()

    return

```

## Sound Playing Module: ColorSounds\_Final.py

```
# =====  
# Module for Sound Files for Listening to Light MQP  
# Authors: Ryan Lang and Amanda Varrichione  
# =====  
  
from time import sleep  
import pygame as s  
  
s.mixer.init()  
  
# These functions play wav files for different program prompts  
  
def backgroundSound():  
    s.mixer.music.load('/home/pi/MQP/SoundFiles/background.wav')  
    s.mixer.music.play()  
    while s.mixer.music.get_busy() == True:  
        continue  
  
def startupSound():  
    s.mixer.music.load('/home/pi/MQP/SoundFiles/startup.wav')  
    s.mixer.music.play()  
    while s.mixer.music.get_busy() == True:  
        continue  
  
def readySound():  
    s.mixer.music.load('/home/pi/MQP/SoundFiles/ready.wav')  
    s.mixer.music.play()  
    while s.mixer.music.get_busy() == True:  
        continue  
  
# Plays a wav file based on color information  
def Color(color):  
  
    if color == 'M':  
        s.mixer.music.load('/home/pi/MQP/SoundFiles/magenta.wav')  
        s.mixer.music.play()  
        while s.mixer.music.get_busy() == True:  
            continue  
    elif color == 'R':  
        s.mixer.music.load('/home/pi/MQP/SoundFiles/red.wav')  
        s.mixer.music.play()  
        while s.mixer.music.get_busy() == True:  
            continue  
    elif color == 'Y':  
        s.mixer.music.load('/home/pi/MQP/SoundFiles/yellow.wav')
```

```

s.mixer.music.play()
while s.mixer.music.get_busy()== True:
    continue
elif color == 'G':
s.mixer.music.load('/home/pi/MQP/SoundFiles/green.wav')
s.mixer.music.play()
while s.mixer.music.get_busy()== True:
    continue
elif color == 'C':
s.mixer.music.load('/home/pi/MQP/SoundFiles/cyan.wav')
s.mixer.music.play()
while s.mixer.music.get_busy()== True:
    continue
elif color == 'B':
s.mixer.music.load('/home/pi/MQP/SoundFiles/blue.wav')
s.mixer.music.play()
while s.mixer.music.get_busy()== True:
    continue
elif color == 'W':
s.mixer.music.load('/home/pi/MQP/SoundFiles/white.wav')
s.mixer.music.play()
while s.mixer.music.get_busy()== True:
    continue
return

```

## 5.2 Device Operation

One motivation of this MQP is to create a device that could be used by the visually impaired to distinguish colors. This appendix provides instructions on how to operate the device.

To start using the device, we must run our Python program called "ListeningtoLight\_Final." To do so, we must have a power source for the Raspberry Pi, either from a wall outlet or using the portable battery source in the electronic project box. Additionally, to start the program, we must connect the HDMI cord from the RPi to a monitor. Once the RPi is plugged in, it will boot up which takes about a minute.

To start the program, we click the start menu in the top left corner of the screen and select Programming. Under programming we select Python 2 to open an IDLE window. In the IDLE window we can open our program by selecting file and then open and then choosing "ListeningtoLight\_Final."<sup>6</sup> To run this module we select run and then run module. The user will know that the module is running when the "start-up sound" is played. With the program running, we are continually given a list of four options to execute: take a background(4), detect a color(5), learn a color(6), learn a background(7). We make a choice by entering the corresponding number of our desired action on the wireless keypad. The user will know to enter input after hearing the "ready" tone.<sup>7</sup>

**Take a Background** Key 4 takes a background. After entering 4 on the keypad, the user is asked to cover the fiber with a black cloth as signaled by a single beep. Do so, and then press enter. A second beep tells the user that they should aim the fiber at a white cloth and press enter. After this action has been completed and the background has been identified, the program will return the chord corresponding to the identified background.

**Detect a Color** Key 5 detects a color. To detect a color we simply aim the fiber at a colored object and enter 5 on the wireless keypad. The program returns the corresponding color 3-note sequence.

**Learn a Color or Background** To learn a color or background we must be connected to a monitor and a full keyboard (through one of the Raspberry Pi's USB ports). After entering the choice 6 or 7, the user should point the fiber at the object whose color is to be learned (for a background point the fiber at a white card). The program will signal that a RGBC measurement has been recorded with a single beep. Then, the user should input either the color or background being learned. Below is a table of keys that indicate the color or background being learned.

---

<sup>6</sup>we can also open the module under the recent files tab.

<sup>7</sup>The machine sound cues are discussed in Section 3.6.

<b>Keys for Learning Colors and Backgrounds</b>	
<b>Color/Background</b>	<b>Key</b>
White	W
Magenta	M
Red	R
Orange	O
Yellow	Y
Green	G
Cyan	C
Blue	B
Purple	P
Brown	BR
Fluorescent Lamp	F
Compact Fluorescent Bulb	C
LED Bulb	L
Tungsten Bulb	T
Sunlight	S



## References

- [1] About Python. (2015). Retrieved October 15, 2015, from <https://www.python.org/about/>
- [2] Breadboard Jumper Wire. Retrieved December 16, 2015, from <https://www.uxcell.com/breadboard-jumper-wires>
- [3] BUD Industries Utility Box. (2016). Retrieved March 01, 2016, from [http://www.amazon.com/gp/product/B005T5A28G?psc=1&redirect=true&ref\\_=od\\_aui\\_detailpages00](http://www.amazon.com/gp/product/B005T5A28G?psc=1&redirect=true&ref_=od_aui_detailpages00)
- [4] Convex Lenses. (2011). Retrieved October 15, 2015, from <http://www.a-levelphysicstutor.com/optics-convx-lens.php>
- [5] Cunningham, C. (2015, May 5). PianoGlove. Retrieved October 15, 2015, from <https://learn.adafruit.com/pianoglove>
- [6] Davidson, M. (2003, March 6). Sources of Visible Light. Retrieved October 9, 2015.
- [7] Demetrius T. Paris and F. Kenneth Hurd, Basic Electromagnetic Theory, McGraw-Hill, New York 1969, ISBN 0-07-048470-8, pp. 512, 546
- [8] Dichroic Filters. Retrieved October 1, 2015, from [https://www.thorlabs.com/NewGroupPage9.cfm?ObjectGroup\\_ID=986](https://www.thorlabs.com/NewGroupPage9.cfm?ObjectGroup_ID=986)
- [9] Eye Lighting International, Light Spectrum Explained. (2015). Retrieved March 30, 2015, from <http://www.eyelighting.com/resources/lighting-technology-education/general-lighting-basics/light-spectrum/>
- [10] Facts and Figures on Adults with Vision Loss. (2014, May 1). Retrieved October 15, 2015.
- [11] Fixed Lens Mount. Retrieved October 10, 2015, from [http://search.newport.com/?q=\\*&x2=sku&q2=LH-1](http://search.newport.com/?q=*&x2=sku&q2=LH-1)
- [12] Fundamentals of Telephone Communication Systems. Western Electric Company. 1969.
- [13] Hecht, Eugene (2002). Optics. Addison-Wesley. ISBN 0-321-18878-0.
- [14] How Sound Waves Work. (2012). Retrieved October 15, 2015, from <http://www.mediacollege.com/audio/01/sound-waves.html>
- [15] Judd, Deane B.; Wyszecki, Günter (1975). Color in Business, Science and Industry. Wiley Series in Pure and Applied Optics (3rd ed.). New York: Wiley-Interscience. p. 388. ISBN 0-471-45212-2.
- [16] Light Waves. (2015). Retrieved October 15, 2015, from <http://www.bbc.co.uk/education/guides/zq7thyc/revision/6>
- [17] Multimode Optical Fiber. Retrieved October 10, 2015, from <http://www.thorlabs.com/thorproduct.cfm?partnumber=FG200LCC>
- [18] Garton, B. Musical Intervals. Retrieved October 15, 2015, from [http://music.columbia.edu/cmc/MusicAndComputers/chapter1/01\\_03.php](http://music.columbia.edu/cmc/MusicAndComputers/chapter1/01_03.php)
- [19] "Optical Fiber". <http://www.thefoa.org/>. The Fiber Optic Association, Inc. Retrieved 3 Oct. 2015.

- [20] Plastic Hybrid Acylinder Lenses. Retrieved October 9, 2015, from <http://www.edmundoptics.com/optics/optical-lenses/cylinder-lenses/plastic-hybrid-acylinder-lenses/84394/>
- [21] Quimby, R. (2006). Acceptance Angle and Numerical Aperture. In *Photonics and lasers: An introduction* (1st ed., p. 45). Hoboken, New Jersey: Wiley-Interscience.
- [22] Raspberry Pi. Retrieved September 28, 2011, from <https://www.raspberrypi.org>
- [23] Raspberry Pi 2 Model b. Retrieved September 28, 2015, from <http://www.mcmelectronics.com/content/en-US/raspberry-pi>
- [24] RGB Color Sensor with IR Filter and White LED. Retrieved October 11, 2015, from <https://www.adafruit.com/products/1334>
- [25] Spectral Power Distribution. Retrieved October 15, 2015, from <http://www.lrc.rpi.edu/education/learning/terminology/spectralpowerdistribution.asp>
- [26] Transmission curves of many filters for monochrome photography, Schneider, p.1
- [27] USB Powered Speakers. Retrieved December 16, 2015, from <https://adafruit.com/products/1363>
- [28] What Wavelength Goes With a Color? (2011, November 15). Retrieved October 15, 2015, from [http://science-edu.larc.nasa.gov/EDDOCS/Wavelengths\\_for\\_Colors.html](http://science-edu.larc.nasa.gov/EDDOCS/Wavelengths_for_Colors.html)
- [29] Wyszecki, Günther; Stiles, W.S. (1982). *Color Science: Concepts and Methods, Quantitative Data and Formulae* (2nd ed.). New York: Wiley Series in Pure and Applied Optics. ISBN 0-471-02106-7
- [30] Young, H., & Freedman, R. (2004). *Mechanical Waves*. In *Sears and Zemansky's university physics: With modern physics*. (11th ed., p. 496). San Francisco, California: Pearson Addison Wesley.
- [31] Young, T., 1802. Bakerian Lecture: On the Theory of Light and Colours. *Phil. Trans. R. Soc. Lond.* 92:12-48. doi: 10.1098/rstl.1802.000