



Institutt for elektroniske systemer
Institutt for elkraft
Institutt for teknisk kybernetikk

Bacheloroppgave

Oppgavens tittel: Colorophone: Et Bærbart Stereosyn basert Sonifiseringssystem for Synshemmede. Project title: Colorophone: A Stereo Vision based Wearable Sonification System for the Visually Impaired.	Gitt dato: 09.01.2019
	Innlevingsdato: 28.05.2019
	Antall sider/bilag: 100 sider rapport + 31 sider vedlegg
Gruppedeltakere: Mohammed Hassan Alkhaledi – dndqm123@gmail.com – 981 07 076 Jørgen R. Bolli – jorgenrb@stud.ntnu.no – 984 71 665 Magnus Jenssen – magnjen@stud.ntnu.no – 959 28 961 Jon Petter Stokmo – jon.petter.stokmo@neanett.no – 908 00 941	Veileder (navn/email/tlf.): Dominik Osiński: dominik.osinski@ntnu.no Tlf: 925 50 365
Studieretning: Industriell Instrumentering / Elektronikk	Prosjektnummer: E1924
Oppdragsgiver: Norges teknisk-naturvitenskapelige universitet (NTNU)	Kontaktperson hos oppdragsgiver (navn/tlf.): Dominik Osiński

Fritt tilgjengelig ☐

Tilgjengelig etter avtale med oppdragsgiver ☒

Rapporten frigitt etter

Blank page

NORWEGIAN UNIVERSITY OF SCIENCE &
TECHNOLOGY

DEPARTMENT OF ELECTRONIC SYSTEMS

BACHELOR THESIS

PROJECT NUMBER E1924

Colorophone: A Stereo Vision based Wearable Sonification System for the Visually Impaired

Principal and Supervisor

Dominik OSIŃSKI

Mohammed Hassan ALKHALEDI,
Jørgen R. BOLLI,
Magnus JENSSEN,
Jon Petter STOKMO

May 28th, 2019



NTNU

Kunnskap for en bedre verden

Abstract

The purpose of this thesis is to evaluate if a stereo camera can be used with the existing Colorophone system, to accurately estimate color and distance in a physical space. Colorophone performs sonification of the visual information, which is transformed into auditory signals, that can be interpreted aurally. This may give visually impaired access to a sensory substitution device, that lets them experience visual space through sound.

A Stereolabs ZED Mini was selected to be implemented in a newly designed prototype. Through use of computer-aided design, a housing was modelled, then 3D printed. By utilizing the sum of square differences algorithm on pixel data extracted from frames captured by the ZED Mini, a binocular disparity is calculated. Through interpolation, the distance is estimated based on the disparity and a calibrated look-up table.

The prototype seemed to reliably estimate distance within $30cm$ to $300cm$, with an accuracy of 65% within a margin of $5cm$. A blindfolded subject was trained and tested in determining distance from auditory signals, and correctly guessed 80% of distances by a margin of $30cm$.

In conclusion, stereo cameras may be a viable choice for the Colorophone system in accurately estimating distance over short to medium ranges.

Abstrakt

Hensikten med denne avhandlingen er å evaluere om stereo kamera kan bli brukt med Colorophone sitt eksisterende system, for å estimere presise farge og distanse verdier i et fysisk rom. Colorophone utfører sonifisering av den visuelle informasjonen, som transformeres til lydsignaler, og tolkes auditivt. Dette kan gi synshemmede tilgang til et sensorisk substitusjons enhet, som lar de erfare visuelt rom gjennom lyd.

En Stereolabs ZED Mini ble valgt for implementering i en nylig designed prototype. Gjennom bruk av datastyrt design, ble en beholder modellert og 3D printet. Ved å ta i bruk en sum av kvadrerte differanser algoritme på piksel data, ekstrahert fra rammene fanget av ZED Mini, ble en binokulær disparitet utregnet. Gjennom interpolering, ble distansen estimert basert på dispariteten og en kalibrert oppslagstabell.

Prototypen virket å pålitelig estimere distansen innenfor $30cm$ til $300cm$, med en nøyaktighet på 65% innenfor en margin på $5cm$. Et subjekt med bind for øynene ble trent og prøvet i å fastslå distanse fra lydsignaler, og gjettet rett på 80% av distansene med en margin på $30cm$.

For å konkludere, stereo kameraer kan være et levedyktig valg for Colorophone systemet i å nøyaktigt estimere distanse over en kort til middels lengde.

Preface

In this project, the goal is to develop a portable and wearable device, that can translate images to audio to aid the visually impaired in sensing the physical world. We wish to look at different techniques to achieve this and to develop our own product from scratch, while taking inspiration from other projects written on this subject.

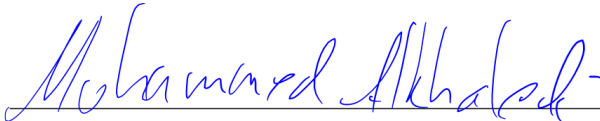
Our thesis is connected to other bachelor projects that our supervisor Dominik has supervised in the past. In 2016, the first prototype of Colorophone was made, and has since been further developed and researched. The first prototype was able to translate colors and distance into auditory signals using simpler, and more limited sensors. The the other projects explored the potential implementation of eye tracking technology to aid usability and control

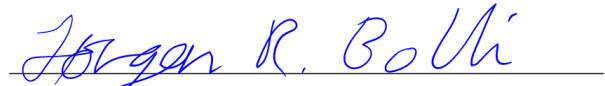
The employer of the bachelor thesis is the *The Norwegian University of Science and Technology*. Our bachelor group consists of three industrial instrumentation students, and one electronics student from NTNU; Mohammed Hassan Alkhaledi, Jørgen R. Bolli, Magnus Jenssen, and Jon Petter Stokmo.


It was decided early on that we wanted to write the thesis in english, as the group thought that it would be easier to write technical terms in english. Writing this in norwegian could have ended up with bad translations of technical terms. Another reason was to make the thesis more accessible, meaning that anyone who is interested has the ability to read it.

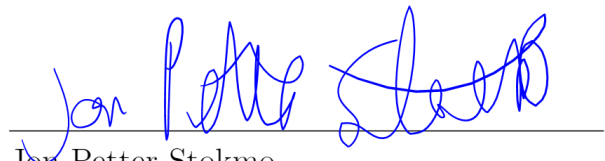
We would like to thank our supervisor, Dominik Osinski for his enthusiasm, guidance, and help during this project.

Date: May 28th, 2019. Location: Trondheim


Mohammed Hassan Alkhaledi


Jørgen R. Bolli


Magnus Jenssen


Jon Petter Stokmo

Contents

Abstract	i
Abstrakt	ii
Preface	iii
Contents	iv
List of Tables	vi
List of Figures	vii
1 Introduction	1
1.1 The project	1
1.2 Colorophone	2
1.3 Further elaboration	3
1.4 Team management	4
1.5 Thesis structure	5
2 Methodology	6
2.1 Knowledge needed	6
2.2 Literary resources	6
2.3 Tools used	7
3 Research	9
3.1 Research objective	9
3.2 Researching stereo cameras	9
3.3 Comparing the D415 and the ZED Mini	12
3.4 Researched hardware that was abandoned	25
4 Theory	26
4.1 Stereo vision triangulation	26
4.2 Stereo cameras	30
5 Implementation	35
5.1 Method for distance calculation	35
5.2 Distance calibration	44

5.3	Software	48
5.4	3D design	58
5.5	3D printing	70
6	The prototype	74
6.1	Optimizing	74
6.2	Testing	77
6.3	Room for improvement	91
6.4	3D- Design and prototype	92
7	Conclusion	94
7.1	Were the requirements satisfied?	94
7.2	Are stereo cameras viable for Colorophone?	95
7.3	Future projects	95
	Bibliography	96
	Definitions	98
	Appendices	101
A	Hardware	101
B	LabVIEW code	107
C	3D-design prototypes	121
D	Additional attachments	128
E	Thesis poster	131

List of Tables

3.3.1	Colors used for testing stereo cameras.	17
6.2.1	Secondary colors as RGB combinations	80
6.2.2	Estimated RGBW values by prototype	81
6.2.3	Distance training exercise results	85
6.2.4	Distance recognition test results	88

List of Figures

1.2.1	Picture: Current <i>Colorophone</i> design	2
3.3.1	Illustration: Mixing primary colors digitally	13
3.3.2	Picture: LCD screen closeup	14
3.3.3	Setup: Color testing stereo camera	15
3.3.4	Graph: Color accuracy, 450lx	16
3.3.5	Illustration: Color scoring equation weakness	16
3.3.6	Graph: Color accuracy, 0lx, weighted	18
3.3.7	Graph: Color accuracy, 450lx, weighted	19
3.3.8	Graph: Color accuracy, 1400lx, weighted	19
3.3.9	Setup: Distance measuring of stereo camera	20
3.3.10	Graph: Distance measurement during direct sunlight	21
3.3.11	Graph: Distance measurement at normal light conditions	22
3.3.12	Graph: Distance measurement in a dark room, high-res	23
3.3.13	Graph: Distance measurement in a dark room, low-res	24
4.1.1	Illustration: Triangulation trigonometry	27
4.1.2	Illustration: Human eye	28
4.1.3	Illustration: Cross-correlation	30
4.1.4	Illustration: Sum-of-squared-differences	30
4.2.1	Illustration: Triangulation disparity	32
4.2.2	Picture: Intel RealSense IR pattern	33
4.2.3	Picture: Intel RealSense IR estimated distance	33
4.2.4	Illustration: ZED SDK 32-bit depth map	34
5.1.1	Illustration: Distance method input data	35
5.1.2	Illustration: Distance method input data	37
5.1.3	Illustration: Best case for prototype	37
5.1.4	Illustration: Best case reference and test data	38
5.1.5	Illustration: Best case Sum.Sq.Diff. grayscale	38
5.1.6	Illustration: Sum.Sq.Diff. method	39
5.1.7	Illustration: Normal case for prototype	39
5.1.8	Illustration: Normal case reference and test data	40
5.1.9	Illustration: Normal case Sum.Sq.Diff. method	40
5.1.10	Illustration: Disparity filtering procedure	41
5.1.11	Graph: Disparity for each horizontal line	41

5.1.12	Illustration: Normal case stereo camera frame	42
5.1.13	Illustration: Worst case for prototype	42
5.1.14	Illustration: Worst case reference and test data	43
5.1.15	Illustration: Worst case Sum.Sq.Diff. method	43
5.1.16	Illustration: Worst case stereo camera frame	43
5.2.1	Illustration: Calibration image	45
5.2.2	Graph: Calibration data for 4416x1242 res	46
5.2.3	Graph: Calibration data for 1344x376 res	46
5.3.1	Illustration: Prototype software communication	49
5.3.2	Picture: Front panel for users	51
5.3.3	Picture: Front panel for developers	52
5.3.4	Illustration: Distance calculation architecture	55
5.3.5	Illustration: Color extraction	55
5.3.6	Illustration: Repeated tick spurs	58
5.4.1	Picture: Aftershokz bone-conductive earbuds	59
5.4.2	Picture: Lego injection mold	60
5.4.3	Picture: Zed Mini internals	62
5.4.4	Illustration: 3D model replica of ZED Mini	63
5.4.5	Illustration: ZED Mini infront of head	64
5.4.6	Illustration: Prototype comparisons	64
5.4.7	Illustration: Final prototype housing design	65
5.4.8	Illustration: Parts of the design	66
5.4.9	Illustration: Prototype framing	66
5.4.10	Illustration: Prototype temples	67
5.4.11	Illustration: Prototype nose pads	67
5.4.12	Illustration: Temple range	68
5.4.13	Illustration: Sideplate for prototype temples	69
5.4.14	Picture: Prototype strap mechanism	69
5.5.1	Picture: Wanhao Duplicator i3 Plus	71
5.5.2	Illustration: 3D printed tensile strength	72
5.5.3	Picture: Prototype stereo camera housing	72
6.0.1	Picture: The prototype	74
6.1.1	Illustration: Area used to estimate distance	76
6.1.2	Graph: Weighing functions for loudness	77
6.2.1	Picture: Yogurt containers	81
6.2.2	Setup: Color recognition testing	82
6.2.3	Setup: Distance recognition testing	86
6.2.4	Setup: Distance testing from prototype POV	87
6.2.5	Setup: First combined test	89
6.2.6	Setup: Second combined test	90
6.2.7	Setup: Third combined test	90

Chapter 1: Introduction

1.1 The project

The aim of this project is to implement a new prototype that is based on an already existing Colorophone prototype. The new prototype should be able to acquire color and distance information through the use of a stereo camera, and turn that information into sound.

A stereo camera is a camera that simulates the human binocular vision. Two image sensor are used to represent human eyes, one on the left side, and the other on the right side. The cameras produce two images looking at the same scene from two different perspectives

The fundamental requirements for the project, given by the customer to us, are as follows:

1. Compare different technologies used in stereo camera systems
2. Pick one or two optimal stereo cameras that (in descending importance):
 - (a) Can be integrated with LabVIEW.
 - (b) Have good reproduction of color.
 - (c) Is small enough to be integrated into the current wearable Colorophone device.
3. Acquire digital distance information for a defined point by using a stereo camera.

The project has some additional, supplementary requirements as well that are of second priority:

1. Integrate existing Colorophone color coding method into a system that utilizes a stereo camera
2. Expand Colorophone into a system that can provide the user with visual echolocation

1.2 Colorophone

Before we can get into the justifications for the project, and further elaborate on the requirements, it is necessary to give an introduction about what Colorophone is.

The original Colorophone prototype, which will be referred to as a product from now on in order to avoid confusion, is a wearable, head-mounted sensory substitution device (see Figure 1.2.1). It transforms visual information into sound signals, with the purpose of enabling the visually impaired to perceive colors and distance through sound [1].



Figure 1.2.1: The current Colorophone design as of January 2019, consisting of a 3D printed housing which contains an ultrasonic sensor, an RGB camera, and a set of bone conductive headphones.¹

By giving red, green, yellow, blue, and white their own unique identifiers in the form of particular frequencies, a user can learn to recognize each color. The amplitude of each color sound signal varies depending on the intensity of the color, thus increasing the perceivable color spectrum.

The most up-to-date *Colorophone* product uses software made with LabVIEW [2] in order to transform RGB pictures and ultrasonic distance values into sound. It utilizes an RGB color model where each of the base colors are associated with a base sine sound frequency. The blue color is associated with low frequencies, green with middle frequencies, and red with high frequencies, whereas white noise is used to represent the color white. The frequencies are chosen based on studies suggesting what frequencies the average person finds most pleasant, and frequencies that do not interfere with vocal speech. By combining the generated auditory frequencies for each color, an appropriate sound signal containing visual information is played for the user so that it can be interpreted. The main idea behind the sensory substitution is that the brain can adapt into understanding the encoded visual information in the sound through appropriate training.

An estimated distance value is generated with an ultrasonic sensor, where the value is delivered to a computer running the Colorophone software so that it can be encoded into a sound signal. From this distance value, a low-frequency ramp wave is generated, which will emit a sound that should sound like a 'tick' to the user. By linearly modulating the frequency of the wave based on the given distance, the intention is that the user will be able to know the distance to the source based on the wave frequency, where small distances create more ticks

¹Picture taken from: <https://www.colorophone.com/technology>

per second, and longer distances create less ticks per second, similar to a parking assistant system in a car. In addition, if the source is closer than $20cm$, the ticks are turned off so that the user can examine the source undisturbed [1].

The team behind *Colorophone* has also made a mobile application version of the Colorophone system called *CLRF Research* [3]. The mobile app can be found on Google Play, and uses the same encoding methods as the most recent product. There is however no distance sonification, since most smartphones lack the required sensors needed to measure distance.

1.3 Further elaboration

This section will try to further elaborate to the reader the purpose of the project, and how we will go about fulfilling the requirements given by the customer.

1.3.1 Reasoning for doing this project

In the current Colorophone product, the acquirement of distance between the user and an object in front of the device is done with an ultrasonic sensor. The sensor sends a voltage output to indicate the distance, which is estimated based on the ultrasonic waves that are being captured. Unfortunately it has limiting capabilities in terms of object localization, while also being unreliable outside of its small defined range. Because of this, the user needs to look directly at an object by turning their head in order to identify it, and could therefore not properly work in tandem with an eye-tracking system, which is another Colorophone feature of interest that has previously been researched by another team of students at NTNU [4]. Another issue with using the ultrasonic sensor is that it is prone to wave interference, especially in industrious areas with motors and welding. The interference can affect the ultrasonic waves, and thus provide false distance information to the user, making it less reliable.

Stereo cameras have rapidly advanced in the recent years, and are being implemented in an increasingly number of devices for their depth and distance measuring capabilities. These stereo cameras have also shrunk in size, and consequentially become more applicable for use in smaller devices. The availability of cheaper, and more reliable stereo cameras is providing great potential for low cost development of advanced stereo vision systems.

It is easy to understand that it would be exhausting for visually impaired users to navigate a room or an item if they would need to move their head all the time with pinpoint accuracy. This is where the potential of using a stereo camera technology comes into play, with its larger distance range, modifiable focus area, and less ultrasonic interference. It is a natural step in the evolution of Colorophone to employ a stereo camera to provide the user with better distance sensing abilities. A stereo camera can potentially replace all the sensors currently utilized, thus providing a more compact package.

1.3.2 How the project was tackled

To get the project started, we had to find a good stereo camera, and so it will be necessary to do research for finding the best consumer stereo camera. At the end of this phase, a stereo camera will have been selected for our prototype.

The team decided together with the customer that we will design a new housing for the prototype, which needs to be robust and lightweight enough so that it can hold a stereo camera while also being comfortably wearable on the head. In the end, we decided that we will utilize 3D printing technology to physically create the housing.

The stereo camera will be integrated with LabVIEW, as it is a requirement. Limiting our choice of software does limit the range of hardware that can be used. However LabVIEW is a great programming environment, and we have access to all the previous Colorphone code generated with LabVIEW.

When the prototype is finally implemented, a subject will test the prototype so we can conclude if its functionality works as intended. In addition to this, the subject will need to utilize the prototype in a big test where all the functionality of the prototype will be put to use.

1.4 Team management

Since this project will be led by a team of four students and supervised by the customer, proper documentation of the project will be an integral part in planning and avoiding potential pitfalls. This subsection summarizes what utilities, and habits the team will take advantage of with the intent of managing the project efficiently.

The team will use a management system where the project life cycle consists of a given number of periods, where each period lasts two weeks, from the start of the project, to the end.

Every second week on a Wednesday, at the start of a new period, the team will have a meeting with the customer in his office at NTNU unless specified otherwise. The day before these meetings, a meeting agenda will be sent to the customer. Regular team meetings will be held at least twice a week in order to keep the team updated on the status of the project.

Timekeeping during the project will be accomplished by using an online excel sheet on Google drive where each member can manually log their work sessions uninterrupted. The timetable for the project's entire life-cycle is shown in Appendix D.1.

After each customer meeting, the team will send the customer a brief document pertaining to tasks that had been done the previous period, tasks for the new period, time usage, and potentially revised project plans. To aid in visually conveying the time spent during the project duration, an S-curve will be utilized. The S-curve is based on the hours logged in the timetable, and can be seen in Appendix D.2.

For long-term planning, the team will utilize a Gantt diagram that will break the project down into manageable chunks and milestones. It will aid in giving the team something to consult when details overshadow the bigger picture, or if the project takes an unexpected turn. It will be appropriately revised to accommodate for any change of plans. The final revision can be found in Appendix D.3.

1.5 Thesis structure

The thesis for this project is written in a chronological structure, as it makes more sense to lead the reader through the entire project timeline, in an effort to avoid misunderstandings.

It all begins with an introduction in Chapter 1 to orient the reader about the project. In Chapter 2, the reference usage, literature research, as well as tools used, is elaborated on. The next chapter in Chapter 3 goes into detail about the research phase of the project, where consumer stereo cameras on the market were compared against each other, and selected for implementation. Further on to give the reader a better understanding on how these stereo camera works in principle, explanations of relevant theory is brought forth in Chapter 4. Arguably the largest chapter, Section 5, goes into the implementation of the prototype that utilizes a stereo camera of choice. The methods used for testing the prototype, including experimental results and data, are discussed and evaluated in Chapter 6. Finally in Chapter 7, it is concluded how well the prototype performed, its shortcomings, and future research and development that could branch of the work done in this project.

Chapter 2: Methodology

Before going into detail on how we did the research, we must first elaborate on the knowledge that was needed in order for us to make reasonable decisions. We had to fill gaps in our understanding, and for that reason we had to get our hands on relevant literary resources. These resources would then need to be put up to scrutiny before we could make use of them in the project and in this thesis. As with almost any modern engineering project, we also had to make use of software tools for us to develop and test our prototype.

2.1 Knowledge needed

First of all we needed to research stereo cameras, since neither of the members had any particular experience with that kind of equipment. Theory and techniques used with stereo cameras had to be understood, so that we could make proper decisions regarding which stereo camera to implement in our prototype. Secondly, we would need to get familiar with how to do 3D design, as the 3D printed parts for the housing had to be made from the ground up to support the stereo camera that we would implement. Naturally, we had to learn how to make proper use of 3D printing and the plethora of available filaments on the market. Lastly, there was the fact that we needed to develop the prototype in LabVIEW. This meant learning the ins and outs on how LabVIEW does things, in order for us to implement bug-free software that could deliver good performance.

2.2 Literary resources

At the start of the project we were given a lot of papers and articles by the customer, that touched on topics like human perception of visual- and auditory information, Sensory Substitution Device's, and animal echolocation. The team read through the papers and articles to learn, but ultimately we ended up not directly citing them, as the paper written by our customer had already referred to what we believed to be the relevant papers [1].

For information regarding stereo cameras or other relevant topics, we scoured the internet. Although the credibility of websites are not so apparent in nature, we kept our internet citations mostly aimed at corporate websites that produced stereo cameras which we could

potentially make use of in this project. For filling holes in our theory we ended up relying on Wikipedia despite its infamous reputation when it comes to citing it in academic papers.

2.3 Tools used

For the project there were several software tools that had to be utilized for us to do testing and comparisons of stereo cameras, not to mention the tools needed for developing the software used with the prototype. This section gives an introduction to the tools used, and why we had to make use of them.

2.3.1 Fusion 360

Fusion 360 is a CAD software for designing 3D models and was developed by Autodesk. The process starts by designing 2D templates and surfaces that can be stretched and pulled to create the desired 3D component. The software can perform simulations such as stress tests with different materials on the 3D designed models. And so the software was utilized when designing the 3D model for the prototype housing, and for doing stress tests.

2.3.2 Ultimaker CURA

Ultimaker CURA is a software that slices 3D models, meaning that it translates the model into commands that a 3D printer can understand so that it can print the model layer by layer, thus 'slicing' the model into many layers. In addition to slicing the model, the tool also decides how the 3D printing will be oriented. It also determines the speed of the printing process, and the quality of the 3D print. Although there is other similar software available, Ultimaker CURA is free, easy to use, and members of the project had previous experience with it.

2.3.3 LabVIEW 2018

LabVIEW 2018 is an integrated development environment (IDE) that lets developers create their own software through the graphical programming language "G" [2]. LabVIEW is most commonly used with prototyping, industrial automation, data acquisition, and other high level applications. The team had to use this IDE since one of the requirements for the project was to integrate the chosen stereo camera with LabVIEW.

2.3.4 ZED SDK

Software Development Kit for Stereolabs ZED cameras. The SDK contains software and code that is intended to aid the development of systems utilizing a ZED camera. The software included in the SDK was used in conjunction with testing during the research phase, and for capturing frames that would be used to obtain calibration data for the prototype. More specifically, the following software was used:

- **ZED Explorer** was used for capturing camera frames with ZED Mini during research and development.
- **ZED Depth Viewer** was used during research to test the distance measuring abilities of the ZED Mini.
- **ZED Diagnostic** was used to validate that the ZED SDK and ZED Mini was installed and functioning properly.

2.3.5 Intel RealSense SDK

Software Development Kit that is developed for stereo cameras made by Intel. The SDK contains several useful features, such as debugging tools, example codes, and wrappers. The wrappers give developers the ability to make use of Intel RealSense stereo cameras in programming languages like C, Python, and LabVIEW. We used the SDK when testing the Intel D415 camera during the initial testing and comparisons of stereo cameras.

2.3.6 MATLAB

MATLAB is a programming- environment and language used for numerical computations. We used it to create scripts that would provide reference data for the binocular disparity, which was then used as control value to make sure that the implemented distance calculation method was correctly functioning.

Chapter 3: Research

In preparation of selecting a stereo camera that would later be implemented for the project prototype, research had to be done so that bad surprises were less likely to occur during implementation.

3.1 Research objective

The Colorophone product planned for this project, was in the form of head-strapped glasses that would be both lightweight and compact, and would function sufficiently across a well defined range of distances. It also needed to be able to handle varied levels of light intensity reliably. A major objective of the project was to modify the existing Colorophone software so that it works reliably with the stereo camera selected during development, so it was important that it could be integrated with LabVIEW.

3.2 Researching stereo cameras

Stereo cameras are often used to compare relative distances for objects within a certain frame of reference. The Microsoft XBOX for example uses the stereo camera technology in their Kinect to track the users movements and positioning [5]. Tesla however, takes data from their stereo cameras and supplements it with data from several ultrasonic sensors as well, to drive the autopilot software that is currently used in their vehicles [6].

During research there was a creeping realization that locating a single market leader for stereo cameras would prove itself to be difficult, as each company took use of different technologies in order to efficiently extract the most visual data from the camera frames. So a natural consequence to this fact was that a sizeable chunk of the research phase was spent on just cataloging, and comparing numerous stereo cameras based on their advertised features, and the project requirements (see Section 1.1).

It was decided to order the two best stereo cameras so that they could be tested against each other later in the research phase. The following specifications were of primary interest during the comparison of stereo cameras:

- **Resolution.** The implementation planned for this project would only need to focus on a small area of visual data. In the future however, eye tracking implementation would be more reliable with more distance data. A higher resolution would have repercussions though, as it would require more computational power, which would result in a higher power draw, greater cost, and more heat output.
- **Minimum and maximum distance that the stereo camera could measure.** A large range would have its merits, but a working range of 10cm to 10m would in principle be way better than 1m to 30m, as a visually impaired user would in most cases analyze objects that were the closest for the sake of navigation.
- **Minimum and maximum light levels that the stereo camera could detect.** This was importance because with the Colorophone, the user would want to use it both inside and outside, and so a large light range was an aspiration of value.
- **Size and weight.** A more compact solution would be easier to integrate with a wearable product design, and a lighter pair of glasses would be more comfortable to wear for longer periods of time.
- **Power efficiency.** A stereo camera with better power efficiency has a lower heat output, which in turn implies that the product does not need a large battery in order for it to have a long-lasting operational time.
- **Cost.** A larger percentile of the visually impaired on earth live in impoverished areas of the world with sub-optimal healthcare solutions, so a lower cost would result in both a higher profit margin through both lower product cost, and a larger market size.
- **Existing SDK available for stereo camera.** If less time was needed in order to implement elementary functions for distance calculations, then more time could be spent on developing more complex functions.
- **Product support and user feedback.** By reviewing user feedback given on a product platform such as a forum, the team can get a glimpse of what real-world capabilities a 3D camera possesses, while at the same time elaborating on features that might not be mentioned in the product spec sheets. This could give pointer on how honest the advertisements of the product made by the marketing departments are.

During the first two weeks of the project the team researched a total of ten different stereo cameras, with each camera having to comply with the project requirements and the specification summed up earlier in this section. Here is a list of the stereo cameras that was not ordered, and why:

- **Intel RealSense R200** [7]. This stereo camera fell short as it was roughly two and half years old during the project's run, and Intel had stopped supporting the SDK used with the R200. The Z300 had replaced the R200 just months after its introduction, and the most recent stereo cameras released by Intel was the D415, and the D435. So it decided to investigate the latest D-series of stereo cameras released by Intel, but not the R200.

- **Microsoft Xbox Kinect** [8]. Microsoft was the first company to introduce stereo cameras to the consumer market. Where previous applications of stereo cameras had been limited to the industrial market, the Kinect brought forth motion tracking into the world of video games. Unfortunately, the newest version, Kinect v2, was too impractical for the project because of its size and power consumption.
- **Orbbec Astra mini** [9]. This stereo camera came in two version, a short-range version with a maximum range of just 1m, and a long range version with a minimum range of 60cm. The range was therefore very limiting compared to other stereo cameras available for developers, so it was decided that it did too poorly compared to the other stereo cameras.
- **E-Con Systems Tara stereo camera** [10]. A stereo camera that lacked good resolution, and had a very limited SDK available, as it was a fairly new startup company at the time.
- **Ensenso N35** [11]. A stereo camera that had many great specifications, due to its high sensitivity sensor and IP67 rated housing. Its sensitivity was advertised to do solid work in detecting low levels of light, and would perform quite reliably during harsh weather. The weight, size, and price was on the other hand outcompeted by Stereolabs and Intel.
- **ASUS XtionPro Live** [12]. It was a stereo camera that was initially marketed as a competitor to Xbox Kinect's stereo camera. It was however challenging to find open source software or an SDK for this product, and its availability was limited since stores required several weeks deliver the product.

With the requirements that was set for the project, and specifications of interest, it was concluded that the best stereo cameras available on the market for this project was being produced by Intel and Stereolabs. Intel's stereo cameras utilized IR projection and stereo vision, whereas the Stereolabs stereo cameras operated with just traditional stereo cameras without any IR projection.

The Intel RealSense D415 [13] and D435 [14] had the best resolution on the market for stereo cameras that used IR projection, and Intel's open source SDK v2.0 had many great features such a facial recognition, and height measurement (if the camera was in a fixed position, and had been calibrated on the spot). See Section 2.3.5 for more on the Intel RealSense SDK. The main difference between the D415 and the D435 was the RGB camera, with the D415 having a rolling shutter, and the D435 using a global shutter. Both also used different housings for the lenses. The D415 had field of view that was 20-degrees more narrow than 90 degree lens used by the D435. The rolling shutter also had bad performance during fast moving paces, but at the same time the D415 was more compact, costs less, and performs equally well in most situations when compared to the D435. And so the D415 was ordered in preparation of the tests that would be done on it and the chosen Stereolabs stereo camera.

Stereolabs is a relatively new company, their stereo cameras stands out from others on the market, since no IR projection is utilized. Their initial effort of developing a stereo camera, the ZED [15], was announced as a direct competitor against the motion tracking camera

that Intel RealSense offered. A motion tracking stereo camera would normally be used for tracking people, faces or objects, with the camera resting in a fixed position, and the main intent for the ZED was exactly that. It was therefore understandable that the weight and size of the camera had not been taken into consideration by the company, which was unfortunate for us, as it clashes with our need for a light weight stereo camera. Stereolabs had however designed a new stereo camera, the ZED Mini [16], and promised to aid in augmenting virtual reality headsets, which meant it was light, small, and had the same features as its bigger brother. Both the original ZED, and the ZED Mini have a good image resolutions that could compete with other stereo cameras, and the minimum and maximum operational range of distance is generous. The second stereo camera ordered was thus the ZED Mini, and ended up being Intel’s D415 challenger.

3.3 Comparing the D415 and the ZED Mini

To compare the two 3D cameras, we created several tests to compare both their abilities as conventional RGB cameras and their ability to measure distance. The tests were carried out at different environmental settings, and was done to recreate probable- and critical use cases. Several iterations of these tests had to be done to obtain and maintain consistent data. It is important to note that the tests implemented are not a industry standard, as we do not have the industrial testing equipment needed. Due to the test equipment and test environment, it is safe to assume that it would be difficult to recreate the exact numbers we obtained. However, considering that this is a comparison, it is important to specify that the specific numbers for each camera do not matter. Rather, it is the relation and difference between the performance of each camera. Therefore, if one was to recreate, and redo the tests described in this chapter, the performance of the cameras relative to each other should still stand true.

3.3.1 Measurement hardware

Comparing stereo cameras is no easy job, as they all have their own strengths and weaknesses. In order to make way for more reliable comparisons, we utilized hardware that would let us measure factors that could influence how the stereo cameras captured a scene. There were specifically two factors of importance; brightness, and replication of colors. See Appendix A.1 and A.2.

3.3.2 Color testing

A good camera should have the ability to capture colors at different lighting environments, and represent them as accurately as possible. To test the color accuracy for each camera we set up three different light environments; dark, medium, and very well lit. This is where the light meter previously mentioned came in handy, as it let us measure the light level as often as possible to maintain and secure a fixed test environments for both stereo cameras.

We started by creating a color palette, consisting of eleven colors that were all displayed on a computer screen individually. We captured a frame with both the Intel D415 and the ZED Mini for all these individual colors, then the image data was analyzed, and the stereo cameras were given scores. Both stereo cameras received a score for low, moderate, and bright light performance. The color palette was created with the help of a website¹, which allowed us to enter one RGB number, and the output would be a .PNG image composed entirely of an area containing the same RGB value.

In theory all colors could be recreated by mixing the colors red, green and blue at different ratios. Electronic displays use this method to recreate colors, where each pixel in an image consists of three values; one for red, green, and blue (see Figure 3.3.2. In a 8-bit panel these values can range from 0 to 255, and are referred together as $(value_{RED}, value_{GREEN}, value_{BLUE})$ (see Figure 3.3.1). The first index is the color value for red, second is for green, and the last index represents blue. One can make an analogy for this method, imagine a painter with only three cans of paint, red, green, and blue. If he would mix certain ratios of these colors, he could potentially recreate the majority of visible color spectrum according to the RGB model.

(Red,Green,Blue)
(128,0,0)



Figure 3.3.1: Showing how a dark red color is made by mixing red, green, and blue.

¹w3schools.com

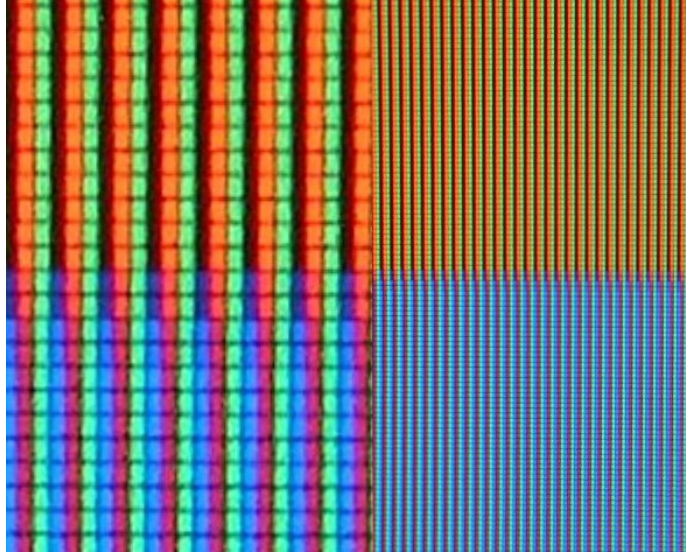


Figure 3.3.2: A closeup picture of an LCD screen, giving a look into how mixed colors are actually portrayed²

In the RGB model used, if a pixel has the values (255, 255, 255), the resulting color would be white, whereas a pixel with the values (0, 0, 0) would be black. So if you wanted a dark red pixel, you could use the value (50, 0, 0). This model can recreate 16.7 million different colors, the display we will be using for our testing however, is a 10-bit panel. A 10-bit value allows for $2^{10} = 1024$ different values for each color, which results in better color accuracy as it allows for a larger color palette of 1.07 billion colors, approximately $2^{(10-8)} = 2^2 = 64$ times increase in number of available colors. But since the ZED mini and Intel D415 both use 8-bit sensors, we will setup our test protocol for 8-bit. This will not affect the end results as our 10-bit display is not the bottleneck, but rather the stereo camera sensors themselves.

It is also important to note that all computer screens display identical RGB values differently due to calibration and hardware limitations. For this test we used the special monitor with 100% sRGB³ coverage, which could recreate input RGB values with incredibly high precision and intensity. We wanted to have the stereo cameras as close to the screen as possible, but within a certain distance they had issues with focusing on the computer screen. After having experimented with several setups, we settled for a distance of 0.5m between the cameras and the computer screen, which can be seen Figure 3.3.3.

²Picture taken from:
https://upload.wikimedia.org/wikipedia/commons/3/34/RGB_pixels.jpg
³standard Red Green Blue: <https://en.wikipedia.org/wiki/sRGB>

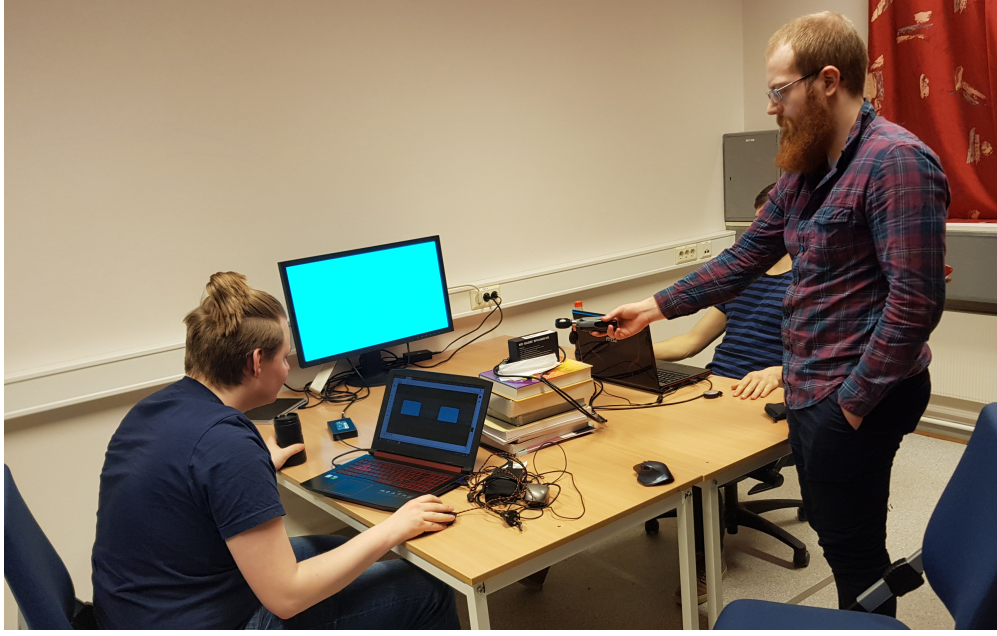


Figure 3.3.3: Testing both stereo cameras by taking pictures of the screen, then later analyzing and comparing the pictures.

After the necessary images from both had been captured, we analyzed the pictures using a website⁴. This website allows the user to import a picture and highlight a small section, then the website outputs the mean RGB value for the pixels selected. To validate this methodology, we manually entered the original color palette itself, and analyzed the data. The website output the same RGB values as the ones initially selected from our palette of eleven colors, thus assuring that it could be relied on.

3.3.3 Analyzing the data for color testing

We entered the mean RGB values that was obtained from all the pictures into Equation 3.3.1, a scoring function we created, indicating the color accuracy to the original.

$$s_{color_acc} = 1 - \frac{1}{765}(\sqrt{(R_R - R_A)^2} + \sqrt{(G_R - G_A)^2} + \sqrt{(B_R - B_A)^2}) \quad (3.3.1)$$

Here s_{color_acc} stands for scoring accuracy. For the colors; A stands for actual, which is the mean RGB color calculated based on a given stereo camera picture, and R is for reference. If the computer display was set to a (255, 255, 255) image, but the camera captured (128, 128, 128), then the score would be 0.5. Higher the score implies better color accuracy. In Figure 3.3.4 one can see how both the ZED Mini and the Intel D415 performed under nominal light quantity of roughly 450lx when utilizing the scoring Equation 3.3.1.

⁴ginifab.com

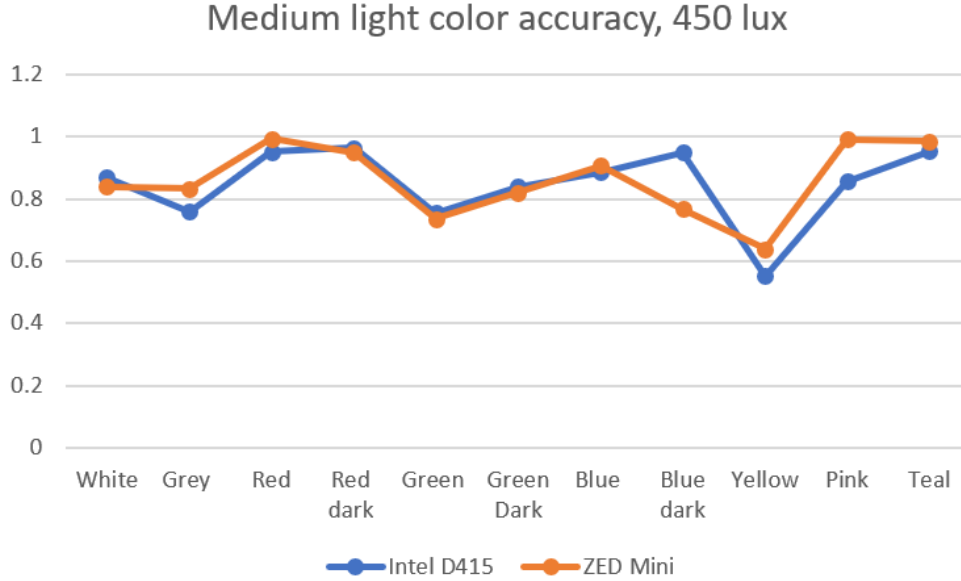


Figure 3.3.4: Color accuracy results in a 450lx environment based on Equation 3.3.1.

We can clearly see that the ZED Mini barely outperformed the Intel D415, as the ZED Mini had an average score of 0.86, whereas the Intel D415 had a score of 0.84. To the naked human eye however, the difference between the captured images for both stereo cameras appeared to be more than the calculated $\frac{0.02}{2}\%$ delta. Further analyzes proved that the error was caused by Equation 3.3.1, because failure to capture the three different colors were punished equally in the overall score.

To give an example, lets look at Figure 3.3.5, the uppermost part is a reference palette (255,0,0) displayed on the computer screen, the part in the middle was captured by a hypothetical stereo camera *A* as (179,0,0), and the lowermost part was captured by another hypothetical stereo camera *B* as (0,179,0). It is highly visible to the human eye that the lowermost part does clearly not match the reference palette at all, whereas the middle part is atleast red. The Equation 3.3.1 however does not handle this situation properly. This is of course a worst-case example created for the sole purpose of highlighting the issue with Equation 3.3.1, with the numbers used not based on actual findings.

In order to create a new equation that could more accurately score the images captured by the stereo cameras, we had to take a different approach. The

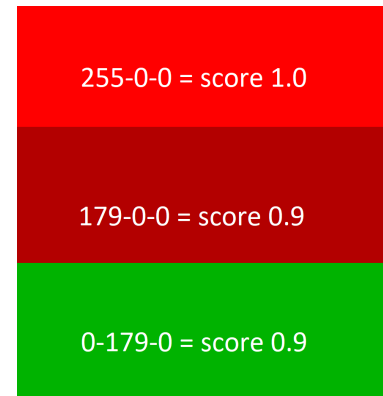


Figure 3.3.5: An illustration showing the problem with Equation 3.3.1 as a consequence of not properly taking other colors into account.

shortfall of Equation 3.3.1 is that a change of 10% will lead to a score that is different by 3.3%, no matter what RGB color component was changed. In Figure 3.3.5 we can still perceive the second box (0, 179, 0) as red, even though red had changed by 30%. It is therefore obvious that a change in red should be considered less important than a change in green or blue, although this is only true when the color red is displayed on the computer display.

Therefore a new scoring system was created, where each color is being multiplied by the values A , B , and C , in which they were selected based on the values seen in Table 3.3.1. A , B , and C are the 'weights' for the three different colors, each color of index x in the color palette has different values for A , B , and C . Sometimes the values for A , B , and C are equal, this is because they are all equal components of the color in question. If we throw colors (179, 0, 0) and (0, 179, 0) from Figure 3.3.5 separately into Equation 3.3.2, the score changes from 0.9 to 0.96 for the mid red, and to 0.87 for the mid green.

Table 3.3.1: *A table of the constants for each color, where the values see use in Equation 3.3.2*

	(R,G,B) Values	x	A	B	C
White	(255,255,255)	1	1	1	1
Grey	(128,128,128)	2	1	1	1
Red	(255,0,0)	3	1	3	3
Red dark	(128,0,0)	4	1	3	3
Green	(0,255,0)	5	3	1	3
Green dark	(0,128,0)	6	3	1	3
Blue	(0,0,255)	7	3	3	1
Blue dark	(0,0,128)	8	3	3	1
Yellow	(255,255,0)	9	1	1	3
Pink	(255,0,255)	10	1	3	1
Teal	(0,255,255)	11	3	1	1

The weights in Table 3.3.1 were chosen after several rounds of testing, as we were unsure at the beginning on how we could calculate, or estimate the values of the weights, therefore we created a visual test. We started with three images of red, with the first one used as a reference of the value (255, 0, 0). For the second picture we changed the red component, and stopped when the image was no longer completely red, but could be described as maroon (140, 0, 0). For the third picture we changed the values for green and blue as well to (255, 47, 0), again to the point where we could no longer describe the image as red. In conclusion, we had to change the value of red by 55% for the second image, but for the second image a change in 18% for green was enough to change the color from red to orange. This ratio of three to one stood true for the other colors as well.

$$s_{color_acc} = 1 - \frac{1}{1785} (A_x \sqrt{(R_R - R_A)^2} + B_x \sqrt{(G_R - G_A)^2} + C_x \sqrt{(B_R - B_A)^2}) \quad (3.3.2)$$

3.3.4 Color accuracy results

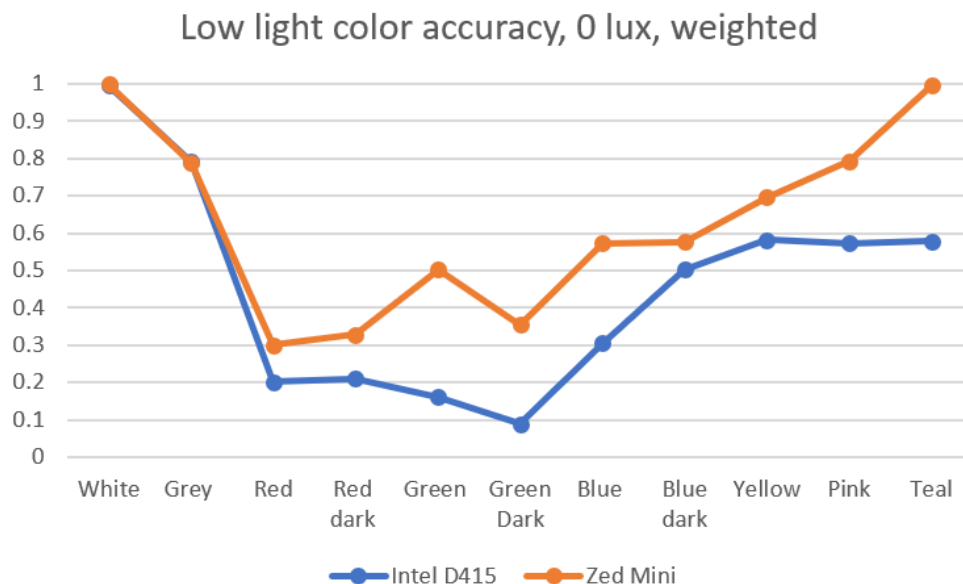


Figure 3.3.6: Graph showing how the Intel D415 and ZED Mini performed in perceiving colors at 0 lux. The values are normalized. At average Intel D415 average was at 0.45, and ZED Mini at 0.63.

In a dark environment the difference was on average 18% in favor for the ZED Mini stereo camera. This specific number is not relevant in a practical sense, as the camera sensor for the Intel D415 is easily overwhelmed in dark environments, where strong and bright colors such as red and yellow are instead perceived to be white. This behavior renders the D415 as an unusable RGB camera during low light situations as implied by the results shown in Figure 3.3.6.

Our results also show that during normal (see Figure 3.3.7), and bright light environment (see Figure 3.3.8) that the difference in color accuracy for both the Intel D415, and the ZED Mini is no larger than the normal test deviation. Both of the stereo cameras performed admirably. It is important to note that the cameras will never have a perfect score because even though the computer screen we are using is calibrated, the rest of the room is not, as the environment had a lot of uncontrollable variables such as light coming in from the covered windows. The lights we used in the room had a yellow and green like tint to it, and we have therefore reason to believe that these lights changed the perceived colors for both cameras. This could also be an explanation for why both of the stereo cameras had a dip in performance for both green and yellow in Figure 3.3.7, and a similar dip for the color red in Figure 3.3.8.

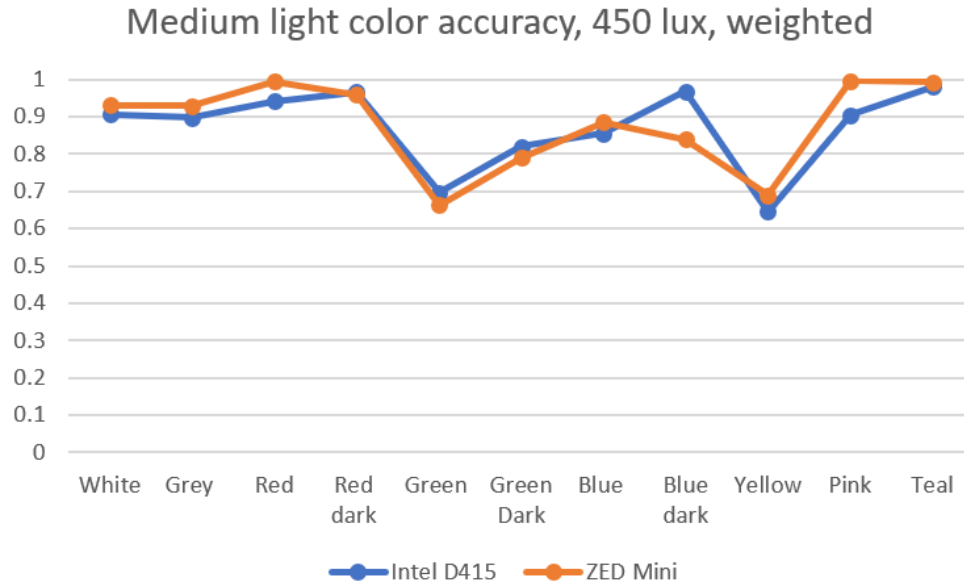


Figure 3.3.7: Graph showing how the Intel D415 and ZED Mini performed in perceiving colors at 450lx. The values are normalized. At average Intel D415 average was at 0.871, and ZED Mini at 0.879.

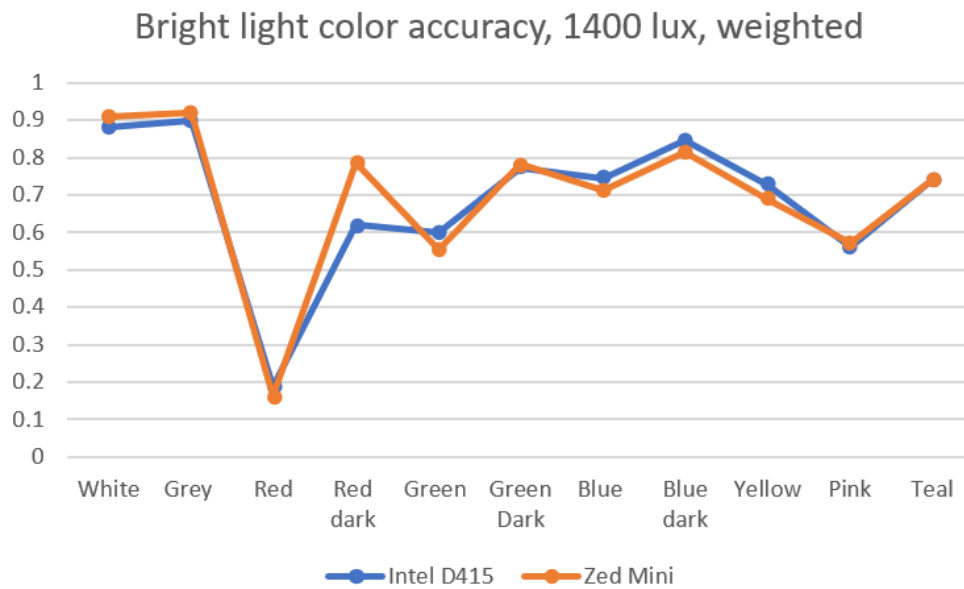


Figure 3.3.8: Graph showing how the Intel D415 and ZED Mini performed in perceiving colors at 1400lx. The values are normalized. At average Intel D415 average was at 0.689, and ZED Mini at 0.695.

3.3.5 Distance measuring

Both stereo cameras have the functionality of calculating distance built in their included SDK. The process of calculating distance for the end user is as simple as pointing the computer mouse at a specific point on the image show in the live stereo camera feed. After our testing procedures were finished, we agreed to create our own software to calculate distance. The software is explored in Section 5.3. This decision did not render the comparisons as useless however, as they showcase the important potential of the stereo cameras.

We wanted to test how accurate and precise the stereo cameras were in measuring distance at different lighting environments. We started by making a simple rig, where both cameras were taped to the cartoon box that the Intel D415 was shipped in. The cameras were then placed so that a measuring tape could measure up to $5m$ without bumping into any walls or objects. If any longer distance was needed, we would mark the $5m$ end, and just move the measuring tape to the new mark, thereby extending the range. Further on we placed a book, covered with white A4 paper, perpendicular to the measuring tape. At first the book was placed very close to the stereo cameras, and then it was moved away from the stereo cameras in increasingly larger intervals. After certain distances we had to increase the size of the intervals to save time. So we began at $10cm$, incremented by $10cm$ up to $100cm$, then went on to increment by $25cm$ up to $200cm$, and so on, until we ended up with measurements ranging at most up to a distance of $10m$. We then stored the distance values given by each camera for all the measured intervals. The tests were done at three different test environments; direct sunshine at roughly $16000lx$, medium light level at $650lx$, and a dark room close to $0lx$, although we suspected that the dark room were not that dark, as the light meter was cheap and thus not particularly accurate.

3.3.6 Analyzing the data, and results for distance measuring

We started by doing this test in direct sunlight, with the stereo cameras pointing in a direction towards the sun. After that we did tests inside in medium light as shown in Figure 3.3.9. Finally we also did tests with dark lights as well.

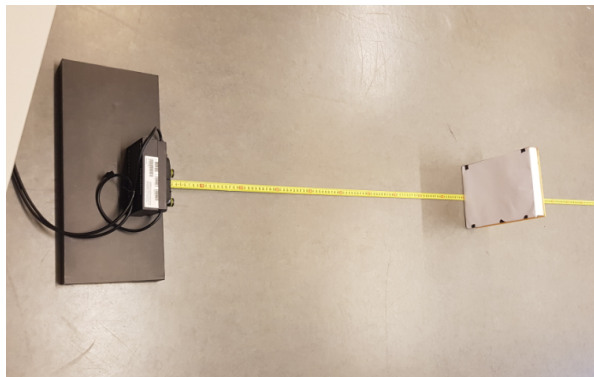


Figure 3.3.9: The setup for distance measuring when testing indoors in medium light.

For orientation purposes; in the resulting distance measurement graphs, the blue line is the actual distance from the stereo camera rig to the reference object, the book. The orange line is the readout from the Intel D415, and the grey line represents the ZED Mini.

It can be seen in Figure 3.3.10 that the ZED Mini does a great job from 20cm to about 2m. The fall in accuracy after 2m could be due to focus issues, as the stereo camera was not able to properly focus on the book. This is nonetheless impressive performance, as we had troubles to focus on the book without shades with the suns bright light coming from behind the book. The Intel D415 on the other hand was unable to return any usable distance readings, and this came as no surprise considering the fact that it utilizes IR technology, which is negatively impacted by sunlight. One behavior that is important to note however, were the nonlinear readouts at close distances for the D415. At 30cm the D415 readout was 70cm, and at 40cm the camera returned to an accurate reading of 40cm. This was a big problem as it made it impossible to determine whether the 70cm was a true reading, or if the actual distance was just 30cm.

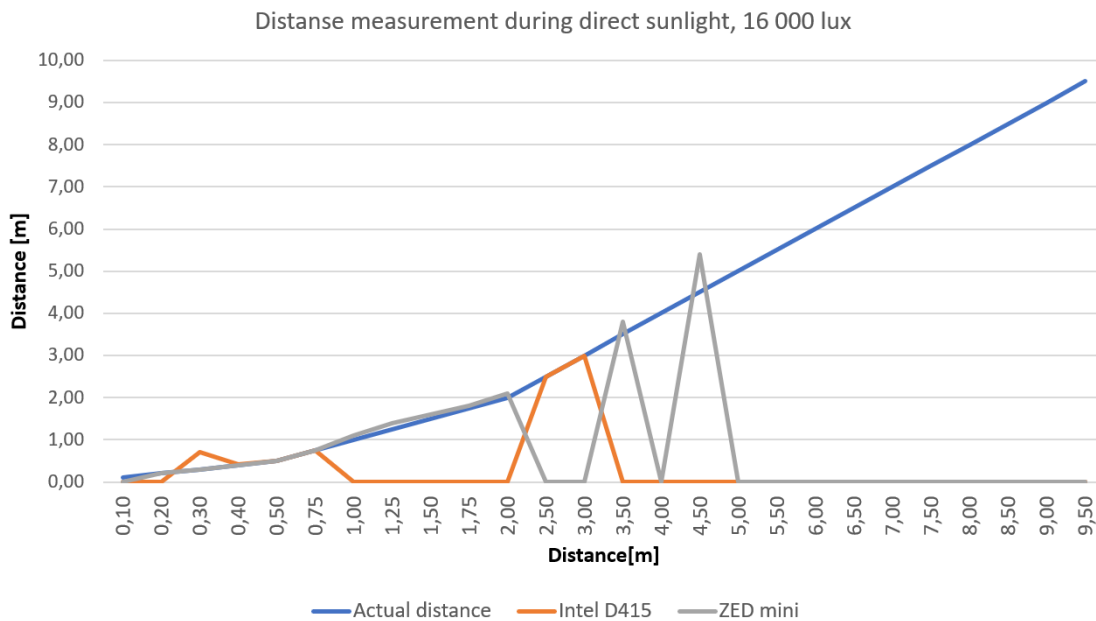


Figure 3.3.10: A graph displaying the stereo cameras measured distances outside with a luminance of 16000lx.

The results shown in Figure 3.3.11 collected during the normal lighting conditions, indicates that nonlinear behavior for the D415 at close distances was still present. The stereo camera performed much better however when compared to the test done outside in bright sunlight. Why the distance calculation suddenly started to deteriorate for the D415 after 3.5m during our initial testing, we were unsure of.

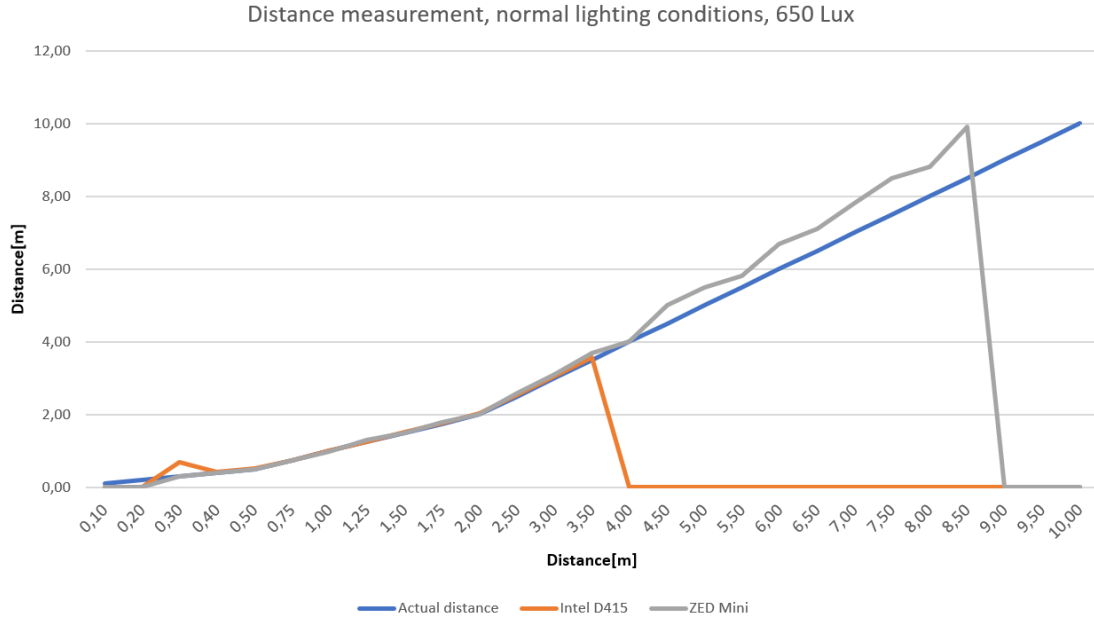


Figure 3.3.11: A graph displaying the stereo cameras measured distances inside with a luminance of 650lx.

One of the benefits of using a IR projector with a stereo camera to measure distance, is how it handles environments with low light conditions, which is why we expected the Intel D415 to hugely outperform the ZED Mini in our dark room test. The results shown in Figure 3.3.12 tells a different story otherwise however, with the D415 actually doing worse than the ZED Mini overall, despite it being a bit more accurate for at certain distances. The D415 still maintains nonlinear behavior in the distance measurements however as can be seen from 3.5m and onwards.

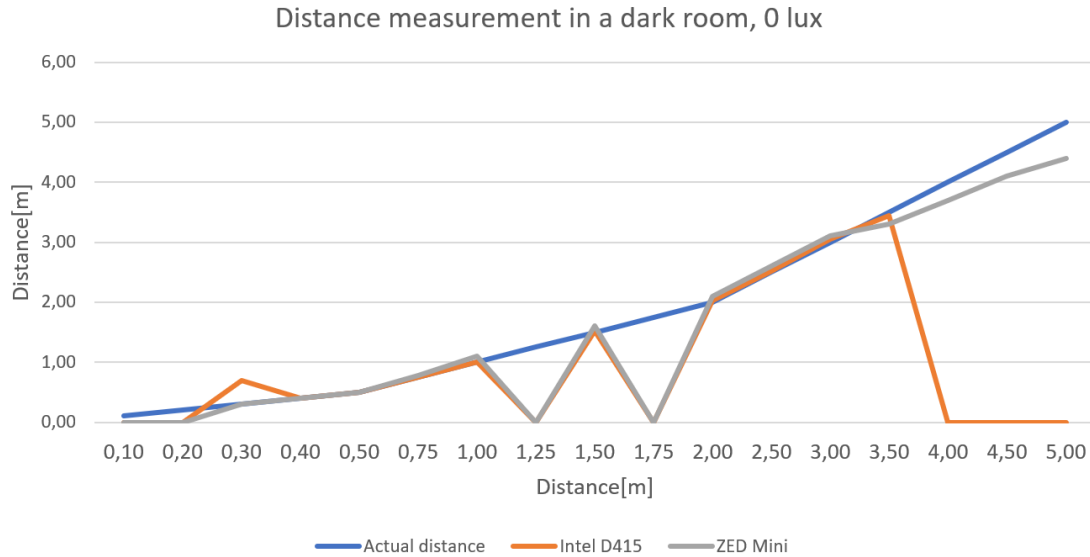


Figure 3.3.12: A graph displaying the stereo cameras measured distances inside with a luminance roughly close to 0lx.

At this point, we realized that something wrong was clearly going on, we performed the dark room test several times and ended up with the same results each time. When we initially received the stereo cameras, we experimented with them and did some simple tests to get a rough idea on how they would perform, and before the actual comparison tests we had experienced that the Intel D415 did a good job in low light conditions. The only difference we actually knew had occurred since the initial testing was that we had upgraded firmware version for the D415. Luckily we did not have just one, but two D415s, with the other not having a upgraded firmware. We did the dark room test again, and noticed there was a significantly better performance this time with the old firmware. After some digging in the Intel RealSense SDK settings, it was found that the only difference between the old and new firmware was that the old firmware had a lower resolution by default. We would thus redo the dark room test with the latest firmware and a lower resolution of 480p instead of 1080p.

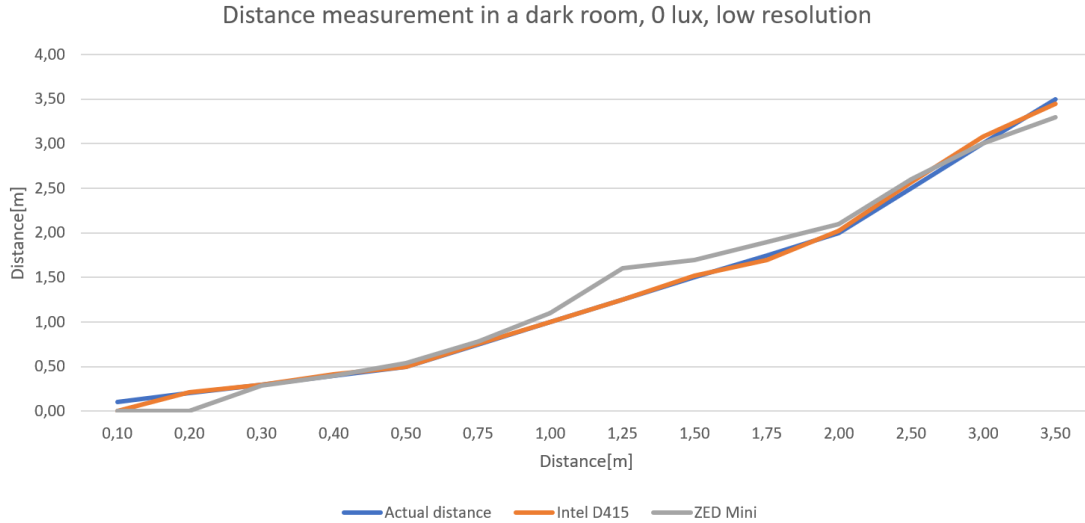


Figure 3.3.13: A graph displaying the stereo cameras measured distances inside with a luminance roughly close to 0lx. For this one however, the Intel RealSense SDK utilized a resolution of 480p with the D415.

As can be seen in Figure 3.3.13, the lower resolution changed the overall performance of the D415 dramatically. The nonlinear behavior stopped, and the camera had better accuracy than the ZED Mini. It is important to note that the latest test with the low resolution was also conducted in a different room. When we tested the D415 in a similar room to the original one with the lower resolution the results changed again. The stereo camera distance readout also became less stable, and we believed this was due to the fact that the highly reflective glass panes in the room would reflect the projected IR pattern by the D415 in a distorted manner.

3.3.7 Stereo camera conclusion

Our results show that in an environment with normal or bright light conditions, the difference in color accuracy for the Intel D415 and Zed Mini is no larger than normal test deviation. However, in a dark environment the RGB camera sensor for the D415 is too easily overwhelmed, with strong and bright colors such as red and yellow being perceived as white, rendering the D415 RGB camera to be unusable in low light situations.

Our initial distance testing suggested that the Intel D415 was useless in measuring distance in both direct sunlight, and in darker environments. With the sensors dialed back to a lower resolution however, it actually did outperform the ZED Mini in darker environments. We decided not to redo the other tests with lower resolutions because of limiting factors with the SDK provided for both stereo cameras.

Our initial plan during the preliminary project was to use the existing SDK provided by the stereo camera manufacturers to calculate distance. Since we intended to use the prototype

as a mobile unit however, this criterion introduced some limiting factors such as limited power draw and computing power. And so during both the color and distance testing it became immediately obvious that the provided SDK were not feasible for the prototype. The provided SDK from both Intel and Stereolabs were difficult to integrate with LabVIEW, especially for the ZED Mini as it required Nvidia Jetson hardware. At the same time, the required computing power was above what could be considered feasible in for a mobile device.

The only option was then to create software from the ground up in LabVIEW that could calculate distance, and doing that with a IR projecting stereo camera such as the Intel D415 was more difficult than with the ZED Mini which utilized stereo vision triangulation. This is due to the lack of documentation defining how the IR technology works in higher detail, thus making it harder to understand compared to regular stereo vision. Combined with the fact that the D415 had bad color accuracy in low light environments, coupled with unstable behavior against highly reflective surfaces, it became obvious which stereo camera would be the better starting point for the Colorophone.

The ZED Mini had better color accuracy for all light levels, and its distance measuring capability was not affected by windows and mirrors. It was easier to develop distance measuring software distance based on stereo vision considering our time frame. The ZED Mini shape and weight also lent itself well to be used for a headset. Based on these findings, we therefore decided to implement the prototype with the ZED Mini in mind.

3.4 Researched hardware that was abandoned

Besides the stereo cameras, other types of hardware were researched as well, specifically hardware that could provide itself as a small computer that could easily be connected with the Colorophone glasses through common connectors, such as USB, in interest of designing a mobile system. This subsection sheds some light on the abandoned hardware that could have had a future in the prototype implementation that we had planned for this project. These are elaborated on in Appendix A.3, A.4, and A.5.

Chapter 4: Theory

The intention of this section is to disclose theory on what a stereo camera is, how stereo camera technology works, and how one can calculate distance based on the data they output. The elaboration on the theory surrounding stereo camera technology is relevant as we rely on it in order to justify decisions made during the project. Before we can get into stereo cameras however, a presentation on stereo vision triangulation is needed in order to explain how two of the more common stereo camera technologies work.

4.1 Stereo vision triangulation

Stereo vision triangulation is a method that is inspired by the human binocular vision system, where one can estimate the distance to a point through the use of two or more cameras and software which can perform triangulation.

4.1.1 Triangulation

For the purpose of explaining how triangulation works at its core, assume that we have a 2D plane where the two points A and B are placed a certain distance L from each other. Draw a line between A and B and call it the baseline. Imagine then that a point P is also located on the 2D plane above A and B such that a line can be drawn from both A and B to P without having the lines intersect with each other or the baseline. Further assume that the angles α and β are known, where α is the angle between the baseline and line going from A to P , and β the angle between the baseline and the line going from B to P . Go on to draw a straight line from P down to the baseline, and define this line as the distance d . You might then end up with a triangle that looks similar to the one shown in Figure 4.1.1.

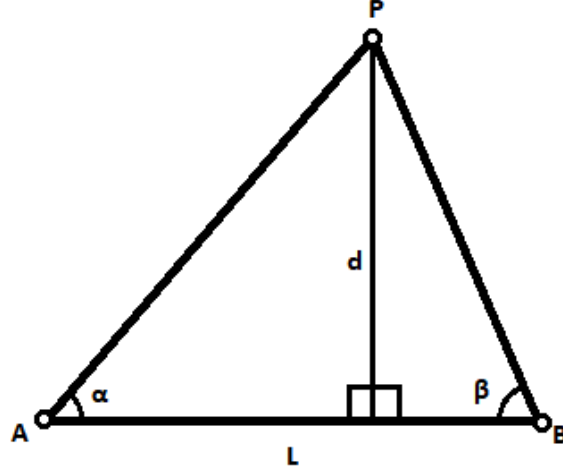


Figure 4.1.1: A triangle formed by the points A , B , and P , where the length L , and the angles α and β are known.

With all this known we can calculate the distance d through the use of trigonometric identities. We can begin by defining the distance $L1$ to be between A to where the vertical line intersects with the baseline. Same can be said for B , whose distance we label $L2$. It then follows that L is the same distance as $L1$ combined with $L2$. Further on, we focus on the two sub-triangles that the entire triangle is composed of. Now from trigonometry we know that $\tan(\text{angle}) = \frac{\text{opposite}}{\text{adjacent}}$, which gives us $\tan(\alpha) = \frac{d}{L1}$ and $\tan(\beta) = \frac{d}{L2}$. We can then make use of this knowledge with the fact that we know $L = L1 + L2$ to find a formula for d . Skipping the algebra process, we end up with the Equation 4.1.1.

$$d = L \frac{\tan(\alpha)\tan(\beta)}{\tan(\alpha) + \tan(\beta)} \quad (4.1.1)$$

4.1.2 Binocular vision system

Humans can understand how far away an object is because we estimate the distance to a point by comparing what our left and right eye sees. If one can imagine that we are hardwired to know our own baseline and focal length, and that we can understand the angle at which both our eyes are looking, the idea is that we utilize some form of triangulation to estimate how far away an object is [17] [18]. The baseline is the distance between the center point of both eyes, and the focal length is the distance from the retina of our eye to the eye lens (see Figure 4.1.2).

Human Eye Anatomy

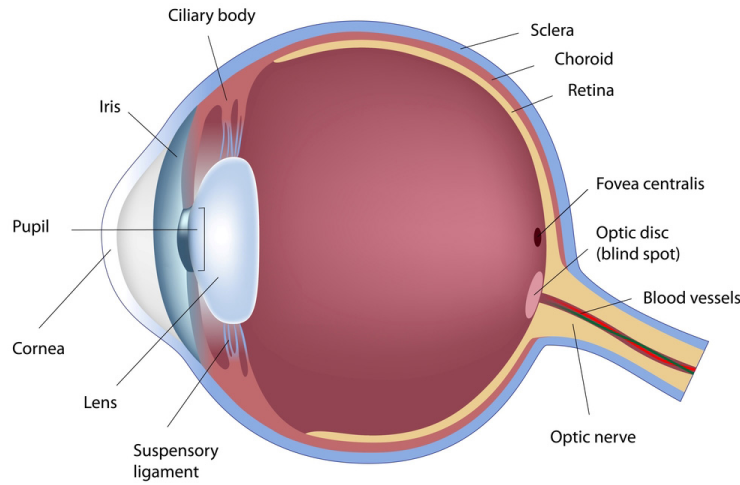


Figure 4.1.2: Illustration of the human eye. ¹

Therefore, one can imagine switching out the eyes in the human binocular vision system with cameras, so that one has a camera for the left eye, and one for the right eye, can be used to estimate distance. There are then a few requirements that must be followed for this to work. First both cameras must be securely placed a certain distance from each other so that the baseline does not change, just like a set of human eyes. Next, the focal length between the camera lens and its light sensor must be known for both cameras, and ideally be equal. What we are left with is a device that can capture images, which can then be used to estimate distance to a point through the use of triangulation.

4.1.3 Binocular disparity

An essential value needed for the distance calculation, in stereo images, is the binocular disparity. The binocular disparity is the difference in location of objects within the right and left image [19].

In the triangulation Equation 4.1.1 presented in Section 4.1.1, the angles α and β are known. When calculating the depth in stereo images however, these angles are initially not known, but could be derived from the binocular disparity in addition to information about the focal length and baseline of the stereo camera.

To calculate the binocular pixel disparity in stereo images, the correlation between pixels from the right and left image must be established. Assume that both the right and left image contains an object that is located in the centre of the left image. The distance to the object is unknown, and therefore it is also unknown which pixels in the right image that displays

¹Picture taken from:
<https://garetina.com/wp-content/uploads/2017/01/human-eye-anatomy-diagram.jpg>

the object. This is what's known as the correspondence problem [20]. The solution to this problem is found using mathematical methods that compare the color values of pixels from the left and right images.

4.1.4 Cross-Correlation and Sum-of-Squared-Differences

Two basic methods that can be used as fundamentals for solving the problem described in Section 4.1.3 is Cross-Correlation and Sum of Squared Differences.

Cross-correlation is the act of measuring the similarity between two numeric sequences as a function of the displacement between them. This is done by shifting one sequence over another while multiplying overlapping values for each iteration. Other names for cross-correlation is sliding-dot-product, and sliding inner-product [21]. Sum of squared differences is a method that adopts a similar way of shifting, but the core equation differs. The latter utilizes the squared difference instead of dot-product, and therefore has the advantage that the input sequences do not need to be normalized around zero, to produce a conclusive output sequence.

Both the cross-correlation and the sum.sq.diff. method can be used to search for patterns in numeric sequences or matrices. The following cross-correlation Equation 4.1.2 and the sum.sq.diff. Equation 4.1.3 can do calculations on two-dimensional matrices. The output data is produced by running the equations a number of iterations corresponding to the size of the input matrices. The start coordinates (x, y) are updated before each iteration.

g : reference matrix	u, v : coordinates in reference matrix
f : test matrix	x, y : start coordinates in test matrix
H : reference matrix height	W : reference matrix width
CC : Cross Correlation	SSD : Sum of Squared Differences

$$CC(x, y) = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} f(x+u, y+v) * g(u, v) \quad (4.1.2)$$

$$SSD(x, y) = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} [f(x+u, y+v) - g(u, v)]^2 \quad (4.1.3)$$

For the purpose of this description, the two input matrices are called the reference-matrix g , and the test-matrix f . When searching for a pattern within test-matrix f , using either Equation 4.1.2 or Equation 4.1.3, the reference-matrix g contains the pattern to be searched for. Reference-matrix g is computed against the test-matrix f . Matrix g is typically smaller than matrix f . The placement of matrix g is shifted over matrix f . By using cross-correlation (Equation 4.1.2), the values in matrix g are individually multiplied by overlaying values in matrix f . When the matrices match, the sum of the multiplied values will peak, and thereby the locations of the matches can be read from the output matrix. On the other hand, by

using the sum of squared differences method (Equation 4.1.3), the matches are indicated by the minimal values in the output matrix.

To further demonstrate the methods in a simpler manner Figure 4.1.3 and Figure 4.1.4 illustrates the two methods with one-dimensional numeric sequences, instead of matrices, as inputs. The yellow reference is shifted over the blue test, from right to left. The number corresponding to the match iteration, in the output sequence, is marked with green. As you can see, the shift needed to match the two sequences can be established from the location of the number indicating a match in the output sequence. In Figure 4.1.3 the match is indicated by the highest value in the sequence. This shows that the cross correlation works well when the input test sequence is normalized around zero. However, if the values in the sequences are all positive or all negative, the possibility for a false match is significantly increased. The sum.sq.diff. method on the other hand, can handle numeric sequences regardless of the normalization. As seen in Figure 4.1.4, the test-sub-sequence most similar to the reference-sequence will always produce the lowest number.

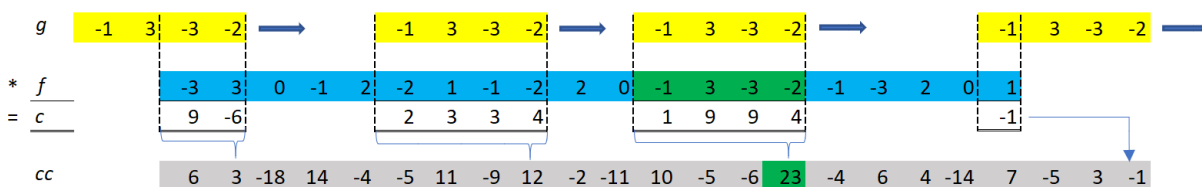


Figure 4.1.3: *Cross-correlation of two numeric sequences.*

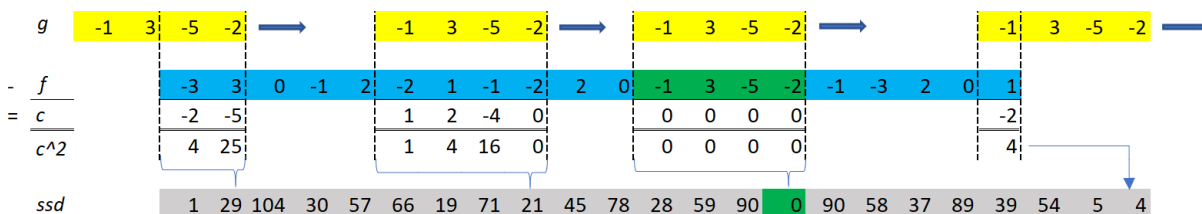


Figure 4.1.4: *Sum-of-squared-differences of two numeric sequences.*

4.2 Stereo cameras

The stereo camera systems that were relevant for this project, outputs image data in the form of two images. One image is a traditional RGB image, whereas the other is an image containing distance data of the scene that the stereo camera is recording. The distance image can then either be portrayed in the form of colors or in grayscale. Even though the output of each stereo camera system looks similar, the software powering the stereo cameras could employ different technologies to generate the distance output.

Among the most common techniques are: passive stereo triangulation, Time of Flight (or ToF), and IR stereo triangulation. The team did however choose not to obsess over what technique would be the best pick, since focusing on selecting a consumer stereo camera that would fit the requirements was first priority, and anything else came second. See Section 3.2 on how the selection process of stereo cameras transpired. From the research done on a wide array of stereo cameras on the consumer market, the following two stereo cameras were selected for further testing:

1. D415 by Intel RealSense, which used IR stereo triangulation technique.
2. ZED Mini by Stereolabs, which used passive stereo triangulation technique.

Since these two stereo cameras were tested and compared against each other, it is of relevance to go deeper into the techniques utilized by both of them. However, an exact description of the technology used in the Stereolabs ZED Mini has been proven to be difficult to obtain, as the algorithms and methods used by Stereolabs are not open source.

4.2.1 Intel RealSense IR stereo triangulation

Intel RealSense uses stereo vision triangulation to measure distance (see Section 4.1), where they employ their own manufactured CPUs to do most of the calculations. On the D415 stereo camera, there are two IR cameras, one RGB camera, and an IR laser projector. The most basic data acquisition of distance can be realized with just the two IR cameras as illustrated in Figure 4.2.1. The disparity value between the center of the left IR camera, and where the object is perceived to be in the viewpoint of the right IR camera, can be used to estimate a distance value between the object and the stereo camera. [22]

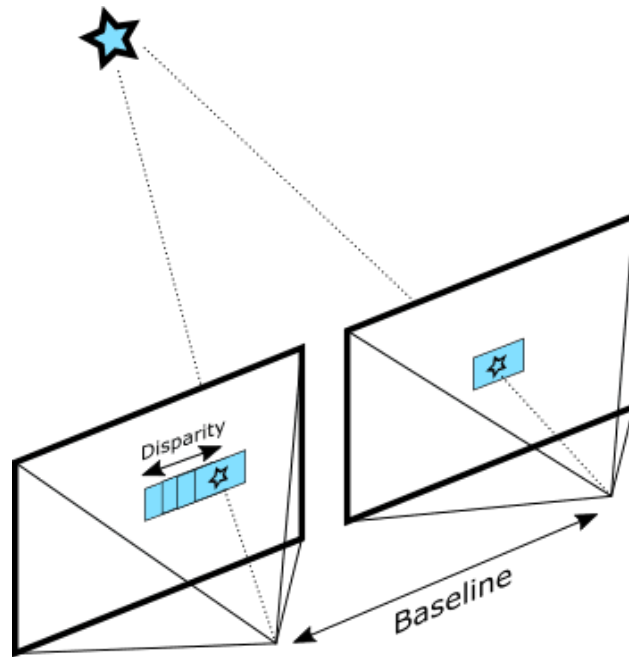


Figure 4.2.1: Illustration on how triangulation is used with the two IR cameras in order to calculate a disparity value, which is then used to estimate the distance to an object or a point.²

This simple system has some limitations however, which is why Intel chose to augment the data by supplementing their stereo cameras with an IR projector. Imagine for a second that the object seen in Figure 4.2.1 is placed in front of a background that has the same color and texture as the object. This will lead to a situation where the algorithm has difficulty calculating the correct disparity value, since it can not distinguish between the object and the background. This problem can be circumvented by utilizing the IR projector to broadcast a unique texture (see Figure 4.2.2), which the algorithm can then take advantage of so that the estimated disparity values are more accurate. It is important to note that this method has proven itself to be quite effective in darker environments, where there is usually little existing IR radiation, which lowers the amount of interference that the IR pattern will be exposed to. It is shown in Figure 4.2.3 how big of an effect a proper usage of the IR pattern can have on the resulting distances.

²Picture taken from:
<https://www.intelrealsense.com/wp-content/uploads/2018/12/stereo-ssd-1.png>

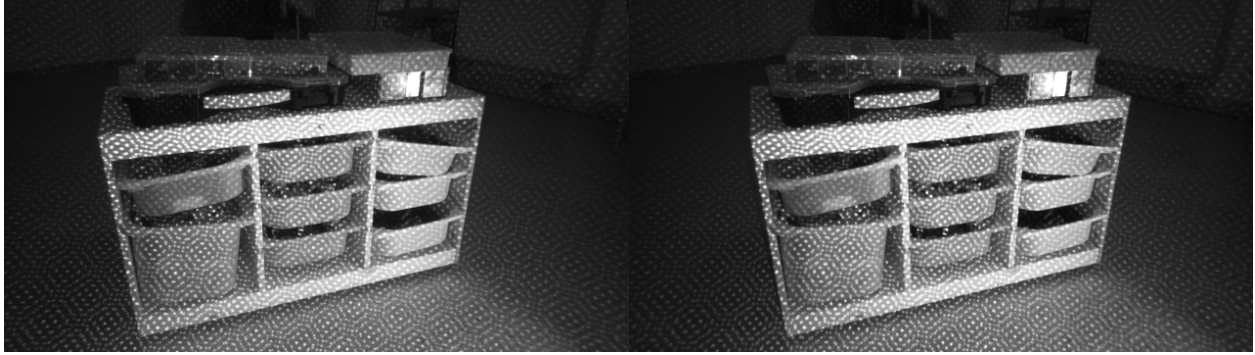


Figure 4.2.2: A monochrome frame that exhibits what the IR pattern projected by the IR projected looks like in eyes of the Intel RealSense IR cameras.³

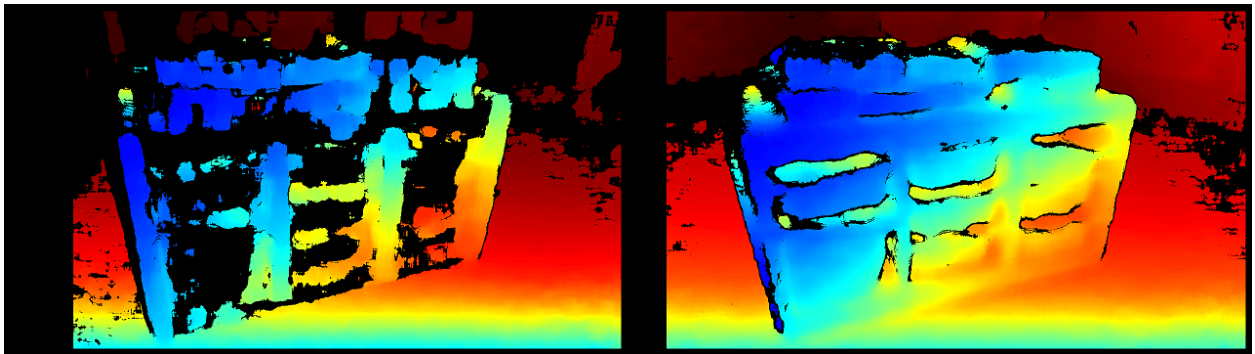


Figure 4.2.3: A multicolored display presenting what effect is had on the estimated distance when the IR pattern is being taken advantage of in the algorithm of Intel RealSense. Notice how the rightmost picture solves the issue of ambiguous distance that is so prevalent in the leftmost picture where the IR projector is not used.⁴

4.2.2 Stereolabs passive triangulation

Stereolabs also base their depth sensing technique on stereo vision triangulation. What Stereolabs does so differently from Intel however, is that they use passive triangulation method, where only two RGB cameras are utilized. Stereolabs SDK makes use of NVIDIA's standalone GPUs in order to do the computations fast. See Section 2.3.4 for more on the Stereolabs ZED SDK.

The algorithm used in the ZED SDK utilizes a depth map that stores the distance value. These maps are encoded in 32-bit, and so the SDK normalizes these values into 8-bit space in the value range $[0, 255]$, which can be shown in a monochrome display. Figure 4.2.4 show

³Picture taken from:
https://realsense.intel.com/wp-content/uploads/sites/63/with_projector.png

⁴Picture taken from:
https://realsense.intel.com/wp-content/uploads/sites/63/projector_effect.png

what this monochrome display looks like in the SDK, where black indicates 0, and white is 255. [23]

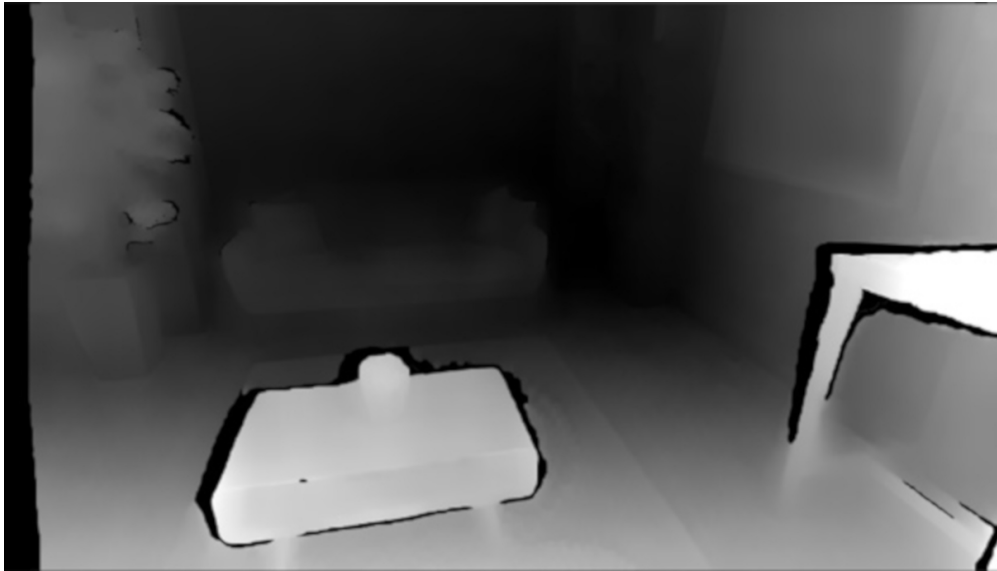


Figure 4.2.4: The 32-bit depth map by the ZED SDK, represented as 8-bit monochrome display.

Chapter 5: Implementation

5.1 Method for distance calculation

Here we go in depth into how the distance calculation method works for the prototype. The method was developed to enable the prototype to function as a sonification system for distance, mainly as a proof of concept. The complexity of the method was therefore kept at a minimum, and it only outputs one single distance value per stereo camera frame. Figure 5.1.1 is a basic model that illustrates the procedure of the distance calculation method. The actual code for the implemented method in LabVIEW is elaborated on in Section 5.3.6.

The process of calculating the distance value is as follows:

- First the **input image data** (see Section 5.1.1) used for the calculation, is obtained from the latest frame captured by the stereo camera, in the form of horizontal lines.
- This data is then processed using the **Sum of Squared Differences** method (see Section 4.1.4); it runs once for every set of horizontal lines (one line from the left frame and one from the right) and outputs one array for each of them.
- Further the **binocular disparity** for each set of lines is found based on the Sum.Sq.Diff. output arrays. The disparity is calculated by subtracting the array-index with the lowest numeric value, from the array length of the Sum.Sq.Diff. array.
- All the disparity values are then **filtered** and combined to one final disparity value (see Section 5.1.4).
- Finally a **look-up table** along with interpolation is used to translate the disparity to distance (see Section 5.1.6).

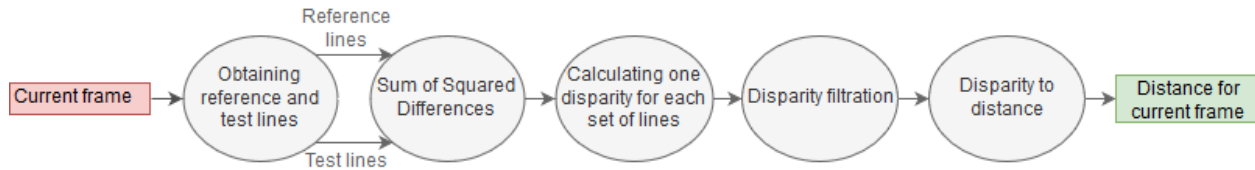


Figure 5.1.1: A basic model of the distance calculation method.

During the development process of the prototype, efforts were directed towards limiting the need for computational power, and that is reflected in how the input data is obtained from each stereo camera frame. The input data must contain enough information to calculate the distance reliably, but also be limited enough so that the less computations must be done. The appropriate amount of input data for the distance function is limited by the information bandwidth of the hearing senses, which has a much smaller information bandwidth than the visual senses [1].

It is important to specify that the examples provided here are all done with the maximum resolution of 4416x1242 provided by the stereo camera. This way more data is collected in each stereo camera frame, which in some cases is an advantage when analyzing the pixel data. The actual prototype runs with a resolution of 1344x376 so that it can handle more frames per second since less resources are needed to estimate the distance for each frame.

5.1.1 Distance calculation input data

The primary aim of the distance method used is to enable the prototype to acquire distance information from a small area in the field of view of the stereo camera. The size of the input data use in calculating the binocular disparity is reduced by extracting a limited number of pixels from the left and right image. The binocular disparity phenomenon manifests itself in the horizontal direction, therefore the image pixels are extracted in horizontal lines. The horizontal lines are $1px$ in height, and are evenly spaced along the vertical axis. The reason for using multiple lines for the computation, is that it will provide a greater precision, which is accomplished by calculating the mean disparity of disparities that has passed through a special filter. The filter is explained in Section 5.1.4.

In Figure 5.1.2 it can be seen that the horizontal lines extracted from the frame have even vertical spacing in between them. This spacing allows for the size of the input data to be reduced, while still covering a broad area of the frame.

For the purpose of keeping things clear, the area marked by the lines in the left image will be referred to as the "reference area", while the area marked in the right image is referred to as the "test area". The pixels used for calculating the distance are obtained from the black lines shown in the figure. The lines of the left image are the "reference lines", and the ones in the right image are the "test lines". The size of the reference and test area, along with the number of horizontal lines in both areas, can be adjusted through a set of defined parameters. These parameters can be accessed in the user interface of the prototype software, and are explained in Section 5.3.5.

The number of extracted lines utilized in the calculations are crucial in reliably determining the distance, but more lines means that more computing power is needed, since more lines must be processed. Therefore the number of lines must be optimized while keeping the relation between computing power and reliability in mind. The process of optimizing this parameter and other parameters relevant for the distance calculation method is reasoned for in Section 6.1.

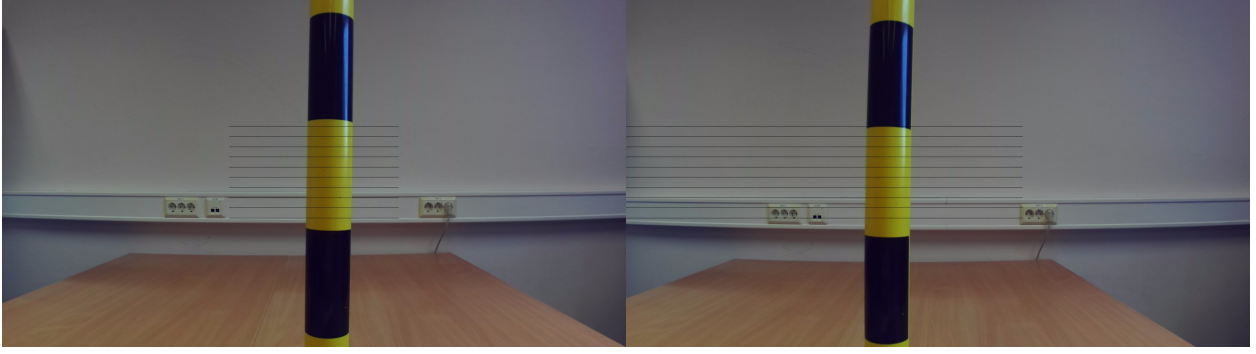


Figure 5.1.2: *Stereo camera frame captured from the ZED Mini. The actual, real distance to the object is 40cm. There are ten lines taken in this case, but might be hard to spot as they are each only 1px in height.*

5.1.2 Best case binocular disparity

The following example demonstrates how the Sum.Sq.Diff. method was implemented in order to calculate the binocular disparity from the reference data and test data. This case acts as a proof of concept in order to demonstrate how the implemented method will respond to what we consider to be a best case scenario.

To better aid in properly illustrating this example, it was preferred to have a large amount of horizontal lines. This was done by setting the spacing between the lines to only 1px, and thereby 160 horizontal lines had to be included satisfy the condition of a defined area height of 320.

The length of the reference lines are set to 600px, and based on this the length of the test lines were calculated to $(4416px / 4) + (600px / 2) = (1404px)$, by using equation 5.1.1.

$$(length_test_lines) = (stereo_image_width/4) + (length_reference_lines/2) \quad (5.1.1)$$

The illustrations can be seen Figure 5.1.3 and Figure 5.1.4.



Figure 5.1.3: *Stereo camera frame captured from the ZED Mini. The actual, real distance to the object is 40cm. The marked area indicates the distance calculation area, where the input data for distance calculation is extracted.*



(a) reference data



(b) test data

Figure 5.1.4: Images composed of the 160 horizontal lines extracted from the black lines seen in stereo camera frame in Figure 5.1.3. The index row for each line goes from top to bottom, so the top line is in row 0, and bottom line is in row 159 (in this case).

The grayscale image shown in Figure 5.1.5 is the output data given by the Sum of Squared Differences equation, described in Section 4.1.4 in the Theory chapter, after having been fed the reference- and test data shown in Figure 5.1.4. The data was square rooted, and then normalized so that the data could be transformed into a grayscale image. The line number is represented along the vertical axis, and the iterations along the horizontal axis.

Each row in the grayscale image has a set of corresponding reference- and test lines with the same vertical placement. Consequently the height of the image in pixels is equal to the number of horizontal lines, 160. The pixels for each row is calculated through iterations of the Sum.Sq.Diff. equation, and indicates the level of color matching between a set of lines.

The darkest spots in the image indicate iterations where the color matching was at its highest. According to this data, the best color matching is in the area around iteration 571. Equation 5.1.2 gives us a disparity of $804 - 571 = 233$.

$$disparity = total_number_of_iterations - iteration_with_highest_match \quad (5.1.2)$$



Figure 5.1.5: Output data after running Sum.Sq.Diff. method with data provided by the best case example.

The number of iterations per set of lines is calculated by the length of the test and reference lines. For the first and the last iteration, the reference lines completely overlaps the test line. Thus the number of iterations is found with Formula 5.1.3. For this particular case, we end up with $1404 - 600 = 804$ iterations.

$$total_number_of_iterations = length_test_lines - length_reference_lines \quad (5.1.3)$$

The Figure 5.1.6 illustrates how the Sum.Sq.Diff method handles the sets of horizontal reference and test lines. One value in the output array corresponds to one iteration.

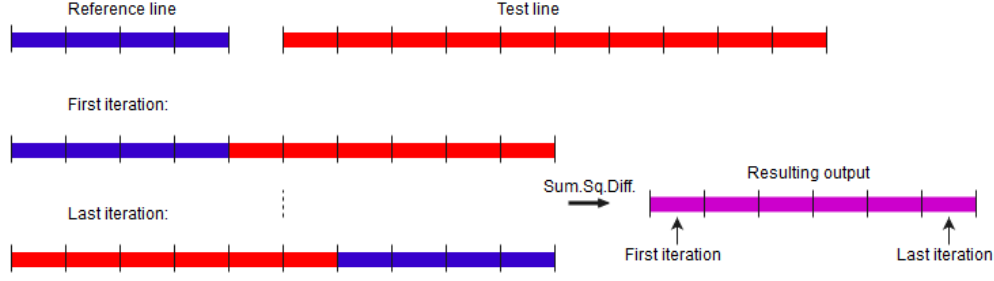


Figure 5.1.6: *Illustration of how the Sum.Sq.Diff method runs the iterations on a set of horizontal lines.*

We have now demonstrated how the method calculates the disparity under optimal conditions. However, under normal circumstances the method is subjected to more difficulty.

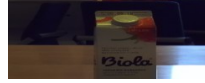
5.1.3 Normal case binocular disparity

When dealing with stereo vision systems, a common problem is that different color perceptions between the left and right lenses of the stereo camera can lead to noise in the stereo camera frames. The following example demonstrates how the method handles normal scenes where multiple different object are captured in the reference- and test data. Just as with the best case example, many horizontal lines over a large height area are extracted so that resulting noise in the output data can be observed.

As can be seen in Figure 5.1.7 and Figure 5.1.8, the stereo camera frame has sub-optimal light conditions, and areas with similar color values are appearing in multiples locations in the extracted reference- and test data sets. Since this will be a rather common situation for most users, it is crucial that we make sure that the system can properly handle cases like this as reliably as possible. Here we utilize the same parameters as the best case example.



Figure 5.1.7: *Stereo camera frame captured from the ZED Mini. The marked area is a set of 160 horizontal lines with a vertical spacing of 1px. The total height covers 320px.*



(a) reference data



(b) test data

Figure 5.1.8: *Images composed of the 160 horizontal lines shown in 5.1.7. Index row goes from top to bottom, so top is 0, and bottom is 159.*

After having calculated the output data using the Sum.Sq.Diff. method just like we did with the best case, we get the grayscale image shown in Figure 5.1.9. Compared to the best case however, this case has dark spots in multiple locations, and not just one area. This 'noise' can lead to false positive matches, and consequently the algorithm will output a distance value that is inaccurate. It is important to note that an inaccurate distance is way worse than an undetermined distance, since there is no way for a presumably visually impaired user to know the difference between an inaccurate and an accurate distance value. If the user experiences a situation with such discrepancy between reality and the estimated distance, then the user will not be able to trust the prototype. In order to avoid this from happening, we implemented a filtration function whose aim is to improve the estimated distance, and only provide the user with a distance if enough reliable data is obtained.

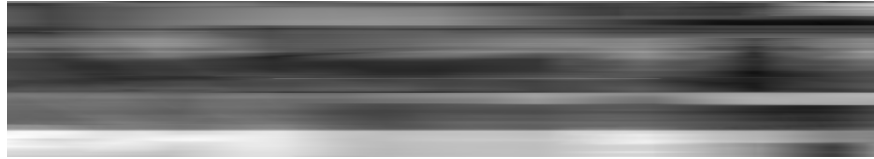


Figure 5.1.9: *This image shows the resulting output data with the normal case example.*

5.1.4 Filtering disparity data

Under normal conditions, some of the extracted horizontal lines will produce a disparity that is correct, whereas the others will not. This is what we will be referring to as 'noise' when talking about disparities. This noise is a problem since it leads to uncertainty about the distance, and the goal of the filtering step is to remove this uncertainty.

The noise in the disparity data often appears as clusters of maximum values, or clusters of minimum values. The first step of the filtration procedure therefore makes sure that only disparities within a defined range are passed on, utilizing a band-pass filter. On the assumption that the noise will be clustered, a mode concept is adopted in the filter, where the most frequency occurring disparities are let through the filter. The mode concept works in tandem with a threshold that makes it so all similar values, within the defined threshold, is included when counting the disparity values. This count is used to determine the most frequently occurring values. A larger threshold value means that the filter will let more disparities through.

Before the final step of the filtration, the function utilizes a `minimum_similar_lines` parameter when deciding if a reliable distance can be estimated, where similar lines is referring to

similar disparities within the threshold. If no reliable distance can be estimated, then no valid distance will be given, which is indicated by returning a distance of -1 . If there is however enough similar disparities given, then a mean disparity is calculated and passed on to the interpolation function. The entire filtration process is illustrated in Figure 5.1.10.

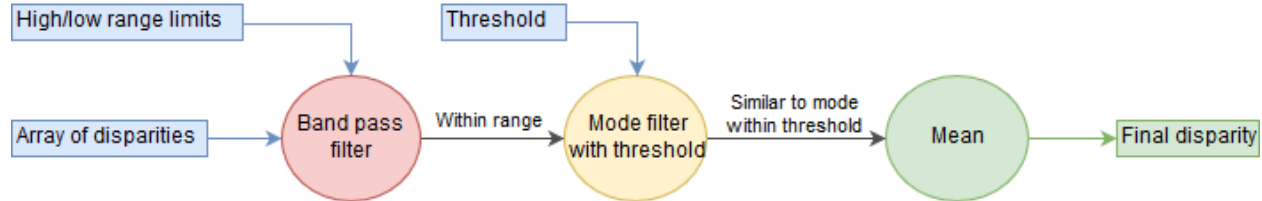


Figure 5.1.10: *Illustration of the disparity filtering method.*

The calculated disparity data from the normal case example is plotted against against the corresponding line numbers, as shown in Figure 5.1.11. Graphs like this can be helpful when determining what threshold value would be the best for cases like this. This graph in particular is well suited for analyzation, as it has a lot of data points due to the large amount of disparities that were calculated off of the 160 horizontal lines.

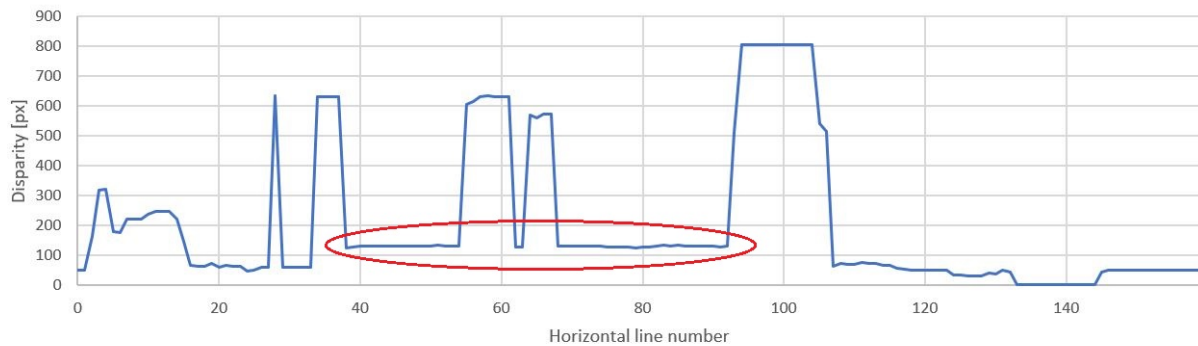


Figure 5.1.11: *Graph of disparity for each horizontal line. It is based on the data shown in Figure 5.1.9*

The disparities surrounding $131px$ within the red circle will be the ones that are passed through the mode filter, since they are the most frequently occurring disparity values. Do notice however that these disparities are not exactly equal, and this is where the threshold parameter comes into play. Consider the disparity value around line number 100, then imagine that the threshold value is set to be really low. In that case, the most frequently occurring disparity will be $800px$ (if this disparity is within range). This is because the disparities within the red circle are not exactly $131px$, and so the smooth line of disparities at $800px$ take priority with such a low threshold. If on the other hand the threshold is set to be too high, then the disparities from line 110 an up might be the most frequent occurring disparities.

Just to confirm that the disparity is $131px$, we measure it by utilizing a MATLAB script. The script shift the colors of the left image to blue, and the right to red. A measuring tool

is used to find the exact disparity, as shown in Figure 5.1.12, which then confirms that the disparity is indeed $131px$.

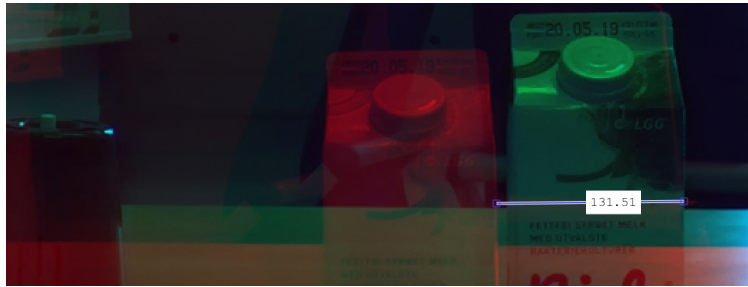


Figure 5.1.12: *The image is composed of the left (blue), and right image (red) taken from the stereo camera frame shown in Figure 5.1.7.*

With that in mind, and the fact that we know $131px$ is the right value, we can estimate a good threshold value. The lowest disparity value in the red circle is at $124px$, and the highest is at $133px$ (ignoring the peaks). This gives us a difference of $(133 - 124)px = 9px$. If we split that in half we get the minimum suitable threshold at about 5 for this particular stereo camera resolution of 4416×1242 , based on the normal case example. In reality the threshold ideally needs to be a little bit higher in order to account for a larger variation within the similar disparities, at the cost of making the mode filter less precise, but more accurate.

5.1.5 Worst case binocular disparity

The following example case demonstrates a potential worst case scenario for the distance calculation method, with the same parameters used in best and normal case. The stereo camera frame seen in Figure 5.1.13 is deliberately captured in such a way that the system is tricked into calculating a disparity that is incorrect. The demonstrated problem is known as the correspondence problem (see Section 4.1.3). It will occur whenever the system is unable to determine what pixels in the test data corresponds to the pixels in the reference data.



Figure 5.1.13: *Stereo camera frame captured from the ZED Mini to illustrate a worst case example.*

It can be seen in Figure 5.1.14b and Figure 5.1.14a that the black boxes are placed in such a way, that the box on the left in the test data is too similar in shape with the right box seen in the reference data. In addition to this, the right box in reference data does not have the same shape in both test- and reference data, making it even harder for the method to identify the correct object.



Figure 5.1.14: *Reference- and test lines taken from Figure 5.1.14.*

The effect of the correspondence problem can be seen in Figure 5.1.15. Notice that we got two areas with similar valleys, meaning that this is essentially a coin flip determines if the disparity is correct or not, and consequently, the same goes for the estimated distance. The distance calculation method will in this case determine that the disparity is $869px$, which corresponds to the darkest area to the left in the figure, and is of course the wrong disparity.



Figure 5.1.15: *This image represents the output data from the implemented Sum.Sq.Diff. method for the worst case example. The data was square rooted, and then normalized, so that it could be translated into a grayscale image. The line number is represented along the vertical axis, and the iterations along the horizontal axis.*

By employing the MATLAB script previously utilized, we end up with a disparity value of $250px$, as shown in Figure 5.1.16. Notice how this disparity represents the rightmost valley shown in Figure 5.1.15.

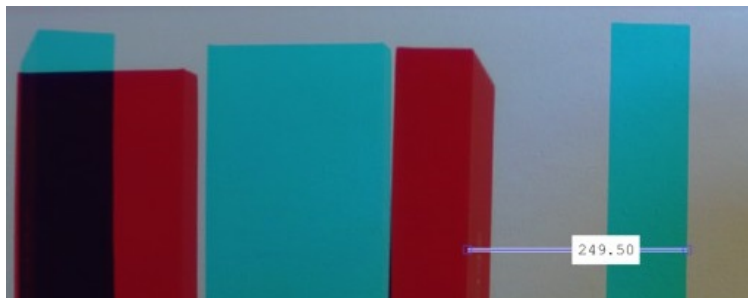


Figure 5.1.16: *The image is composed of the left (blue), and right image (red) taken from the stereo camera frame in Figure 5.1.13.*

5.1.6 Disparity to distance

In Section 4.1.1, the Equation 4.1.1 for calculating the distance d based on the angles α , β and the baseline L was presented. However, the approach for obtaining the distance in the method implemented in the prototype is somewhat different. Since the binocular disparity has already been obtained in the previous steps, only the relation between the binocular disparity and the distance is needed to derive the distance value for the current frame. This relation is obtained from a look-up table, and also through an interpolation between the fixed values in the look-up table, to account for continuous disparity and distance values. The look-up table is generated through a calibration routine.

5.2 Distance calibration

This section goes in depth into how to calibrate the distance function implemented for the Colorophone. If the camera is replaced on the Colorophone, the disparity vs distance relation will change. Consequently a calibration must be performed after having changed to a different stereo camera, or having changed the camera resolution.

To enable the prototype to calculate the correct distance, the values contained in the look-up table must be specifically adapted to the stereo camera and its resolution. A calibration routine must therefore be performed as to generate the look-up table, which is in the form of a text file with the name "calibration.txt", and consists of listed disparity values with corresponding distance values. The text file can then be used by the main Colorophone software.

There is more than one possible method for calibration. The calibration could be done by manually measuring disparity in images and writing the values in the calibration text file. However, to streamline the calibration process, a LabVIEW code is made specifically for the calibration. This code is not implemented into the main Colorophone software because it is regarded to be outside the scope of normal operation.

The calibration code writes the calibration text file with the use of an image series that is captured and saved to disk beforehand. One line in the calibration file is written for each stereo image in the series.

The input images must meet a certain standard to produce a good calibration. It must be stereo images, like the one in Figure 5.2.1, with a solid color target-object contained in the middle of the reference area, that is the centre of the left image. The target-object should extend over the complete height of the reference area and have a width of about 4 to 8 cm. The background, that is to say the area for pixel extraction not covered by the target-object, must be of a solid color different from the target-object. Preferably the target object could black and the background could be white, or the opposite. It is essential that the stereo images is captured with the same resolution as the system would use under normal operation.

To make the calibration easier we used a tripod for taking the pictures, which is elaborated further in Appendix A.6.



Figure 5.2.1: *One of the calibration images, taken at 200cm from a white wall with black tape as the target-object. The frame is captured with resolution 1344x376.*

5.2.1 Procedure

1. Capture the stereo images within the desired range of the system, ranging from lowest to highest distance to target-object. Store all the captured images in the same folder. The file names must start with the distance to object in centimeters. For example: "120.png". If a longer file name is wanted; all the images must start with the corresponding distance and have the same extension after. For example "80cm_ZED.png", "100cm_ZED.png", and so on. The number of images should be eight or more, as it will have an effect on the accuracy of the calibration.
2. Open the "Colorophone_distance_calibration.vi".
3. Select the folder containing the image series, and the folder you wish to save the calibration file in.
4. The settings should be set to the same values they have in the main Colorophone software.
5. Run the "Colorophone_distance_calibration.vi".
6. The new "calibration.txt" file should now be saved in the selected folder. Replace the old "calibration.txt" file, located under the "Files" folder in the Colorophone complex project path, with the new "calibration.txt" file.

5.2.2 Resulting calibration data

The team performed two calibrations, one with maximum resolution and one with minimum resolution. In Figure 5.2.2 and 5.2.3 the calibration data is plotted as graphs. The distance, that the Colorophone would output for each calculated disparity, can be read from the graph.

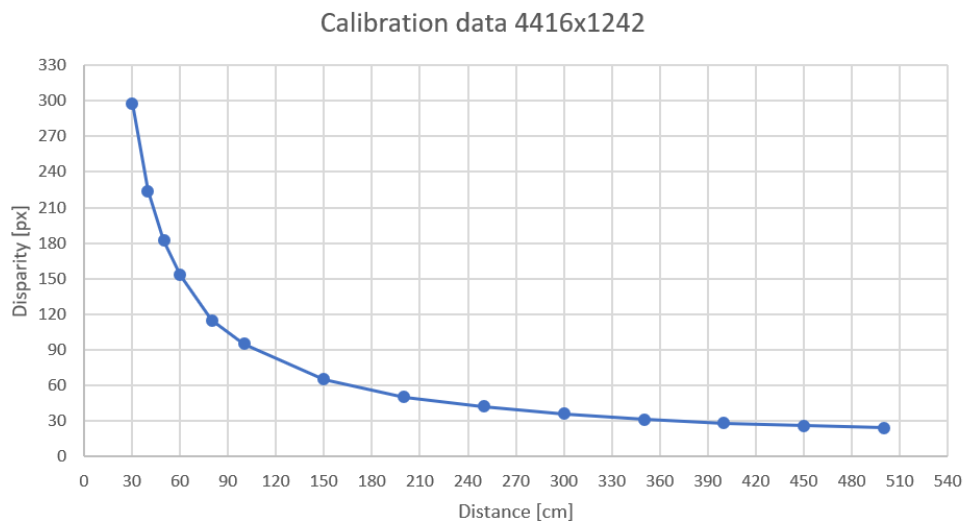


Figure 5.2.2: The graph contains the data obtained from a calibration performed with the use of procedure 5.2.1 and 15 calibration images. The images were captured by the ZED Mini with a resolution of 4416x1242

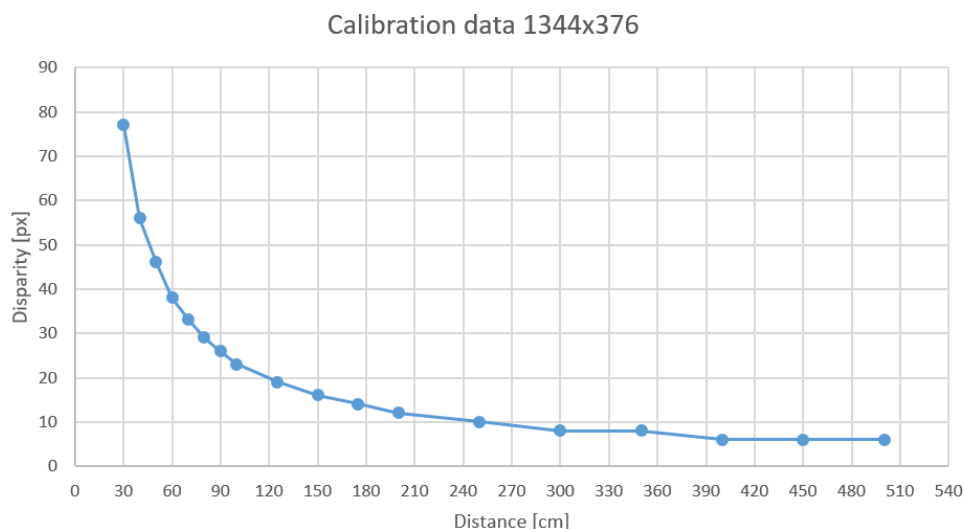


Figure 5.2.3: The graph contains the data obtained from a calibration performed with the use of procedure 5.2.1 and 15 calibration images. The images were captured by the ZED mini with a resolution of 1344x376.

Notice that the image resolution used for the calibration in Figure 5.2.2 is higher than the one used in Figure 5.2.3. As it can be seen in the graphs, the potential for distance precision on longer distances is higher when using a higher image resolution. In the graph in Figure 5.2.3 the disparity is completely flat at a distance of 400cm to 500cm. On the contrary, with the high resolution in Figure 5.2.2, the distance is still possible to derive from the disparity value at this distance. However, the high resolution demands more computing power, and leads to lower frame rates and higher delay in Colorophone software. The 1344x376 resolution as in Figure 5.2.3 is therefore the one implemented in the prototype as default.

5.2.3 Limitations with the method

The distance calculation method was implemented with the intent of it being fairly straightforward, yet functioning. In addition the prototype utilizes the lowest resolution provided by the ZED Mini at 1344x376, so that the capture delay is minimized, and the number of captures per second is maximized. Naturally, these implementation decisions has brought about some limitations to the prototype, more specifically:

- **Trouble handling small objects.** If an object within the reference area takes up less than approximately half of that area height, then the software could end up returning the distance to whatever is behind the object. Thus the object is not detected by the prototype.
- **Static area in field of view.** The method is not prepared to be used for a varying location of the area subjected for distance calculation, within the field of view. Although the size of the area can be adjusted based on the distance calculation parameters.
- **Areas of similar colors.** The system has a tendency to give unstable distance results whenever the same color is covering the whole reference area. This is due to correspondence problems caused by a large amount of pixels that has similar RGB values (see Section 4.1.3, and consequently an incorrect disparity value is calculated.
- **Limited range** The range is somewhat limited due to the low resolution of 1344x376. The high range limit is set to 300cm, and the distance calculation is most accurate at even lower distances than that. However the resolution can be adjusted up in the software, at the dispense of frame rate. Note that a new calibration must be preformed when changing the resolution.

5.3 Software

By utilizing and adapting the current software developed by Colorophone to work with a stereo camera, the team was able to set the foundation for what the next evolution of the Colorophone product might be. The software was developed using LabVIEW 2018, which is elaborated on in Section 2.3.3.

5.3.1 The application

The application itself consists of three parts that will run in the following sequence when **main.vi** (see Figure B.1 in appendix) is executed:

1. *Initialization*
2. *Run*
3. *Close (or Destroy)*

The first part, *Initialization*, declares and initializes data that is necessary for the second part of the application to function properly. This is data in the form of four FIFO queues, five user event queues, a Boolean array containing four state Enums, and a I32 timeout value. See Appendix B.2 for code. The importance and usage of this data is elaborated on in Section 5.3.2.

The second part, *Run*, consists of five processes running in parallel loops, and is where the user is able to interact with the application. More on this in Section 5.3.2.

The third part, *Close*, basically does what the first part does, but in reverse. It is needed because in LabVIEW, queues have to be manually closed in order to properly deallocate memory. See Appendix B.3 for code.

Since the actual application resides within *Run*, from here on out in Section 5.3, *Run* will be referred to as the actual application for the rest of this Section for convenience.

5.3.2 Software architecture

The chosen software architecture for the application utilizes a combination of message- and event-handling queues to make sure that user interaction, the stereo camera, and sound is synchronized. The application starts with the *GUI* process, which resides in **gui.vi**, and is responsible for managing user interactions, and maintaining communication between the four other processes running in parallel; *Clock*, *Camera*, *Sound*, and *Logger*. A simple illustration of this communication between processes is shown in Figure 5.3.1. These processes reside within their own respective .VI's, namely **clock.vi**, **camera.vi**, **sound.vi**, and **logger.vi**. These four processes will be referred to as subprocesses from here on out.

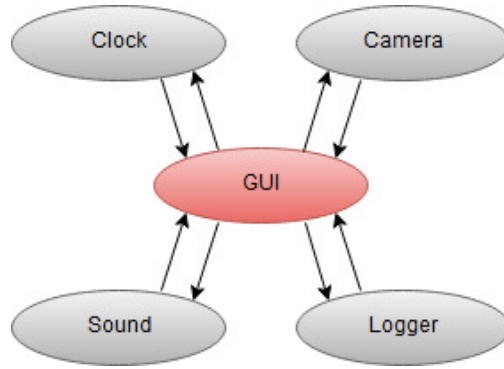


Figure 5.3.1: *Simplified representation of how the subprocesses Clock, Camera, Sound, and Logger communicate through the GUI process.*

In closer detail, *GUI* consists of several steps that prepares the user interface elements and subprocesses. After this it goes into a loop that contains a event structure that can detect new user events. The loop lasts for as long as the program runs, and will close unless an error occurs, or until the user quits the application by pressing the 'QUIT' button.

The *GUI* process communicates with the subprocesses through a *message queue*, which is simply a FIFO queue that is shared between *GUI* and a subprocess, and can hold messages that contains an action and a payload. Each subprocess has one message queue, and *GUI* has access to all of them. An action can be one of the following cases:

- **'Idle'**: subprocess is idle (usually the process does nothing).
- **'Start'**: subprocess starts.
- **'Stop'**: subprocess stops.
- **'Close'**: subprocess closes. Does not close the subprocess loop.
- **'Quit'**: subprocess quits. This closes the subprocess loop.
- **'Settings'**: subprocess updates its settings with new settings contained in the payload.
- **'Read'**: a general action, each subprocess usually acts differently when this is received.

All subprocesses are programmed to react somewhat similarly to these defined actions, except for the 'Read' action, which is more general in nature and used for special functionality like *Sound* playing a sound, or *Logger* logging some data.

Subprocesses communicate with *GUI* through user event message queues, which the event structure in *GUI* can react to and handle accordingly. As with message queues, these are FIFO queues, where each subprocess has access to one that is also shared with *GUI*. These event queues do however not utilize a shared type definition like the message queues, but rather a private one for each subprocess, meaning the developer can easily add new events for one process without giving other subprocesses the same type of available events.

The following events come with the prototype software, some common to all subprocess, and others specific to a certain subprocess:

- **'Null'**: Available for all subprocesses. does nothing, and is never actually used. Simply there to work as a 'filler' for the event message type definition for each subprocess. Technically not needed in the final prototype, but was left anyways as it did no harm.
- **'New_State'**: Available for all subprocesses. Updates the state in *GUI* for the subprocess that sent the event message.
- **'Raw_Images'**: *Camera* subprocess only. Updates *GUI* with the newly captured stereo camera frame.
- **'Sonification'**: *Camera* subprocess only. Gives colors and distance values to *GUI*, which are then sent to *Sound* and *Logger*.
- **'Trigger'**: *Clock* only. Tells *GUI* that *Camera* can capture a frame. Will occur as many times per second as defined by the 'desired_fps' parameter in the application settings. See Section 5.3.4 for more on the application settings.
- **'Clock_Info'**: *Clock* only. Gives *GUI* information that *Clock* possesses such as how many milliseconds it takes for a trigger to occur. Gets sent every second.

In addition to the event- and message queues, the *GUI* process, as well as all the subprocesses have access to another queue that is specifically designed for sending error messages back to *GUI* if something goes wrong.

Each subprocess has a state, these being 'Stopped', 'Running', and 'Quitted'. These will be set based on the action given by *GUI* to a subprocess, and the subprocess state at the time. If for example a subprocess is told to 'Stop' when it is already 'Stopped', it will not do anything. However if it was 'Running', it would stop and change state to 'Stopped'. This is accomplished through the use of case structures. The *GUI* process knows all subprocess current states so it can react accordingly to subprocess user events. When the user presses the 'QUIT' button, *GUI* will send a 'Quit' action to all subprocesses, then they will quit and tell *GUI* that they have a 'Quitted' state. If all subprocesses have the 'Quitted' state, the *GUI* loop will close, and thus the application will close down.

5.3.3 Front panel

The front panel for the application has two modes, one for the developer and one for the user. Developer mode can see and interact with all the same GUI elements as the user mode, but not the other way around. Developer mode can be set making sure the "developer_mode" conditional symbol for the software project in LabVIEW is to "true". It is by default set to "false", meaning it will be in the user mode.

The front panel consists of the following GUI elements across both modes:

- **Image display** that show stereo camera frames, and updates whenever a new frame is received.
- **Quit button** that closes the application when pressed.

- **FPS counter** which indicate the frames per second for the application in integer form, more specifically the amount of stereo camera frames received.
- **Four sliders**, one for each color red, green, and blue, and one for the whiteness. The slider goes from unsigned integer values 0 to 255. For more on how these values are determined, see Section 5.3.7.
- **Color box** which shows what color is the combination of the three primary colors shown in the sliders.
- **Distance indicator**, a text box that tells the user how far away the object in the center of the left image is estimated to be by the software. How this value is calculated is elaborated on in Section 5.1.
- **Distance found?** enables the software to indicate if it has found a trustworthy distance. The indicator is turned on if the distance value is within range, and the "disparity_filtration.vi" defines the value to be trustworthy (see Section 5.3.6).
- **Camera selected.** Gives the user the ability to select what camera to use during run-time.
- **Camera modes.** Lets the user select what camera mode they want to utilize. The available modes are fetched from the selected camera.

A picture of the application in action during user mode can be seen in Figure 5.3.2.

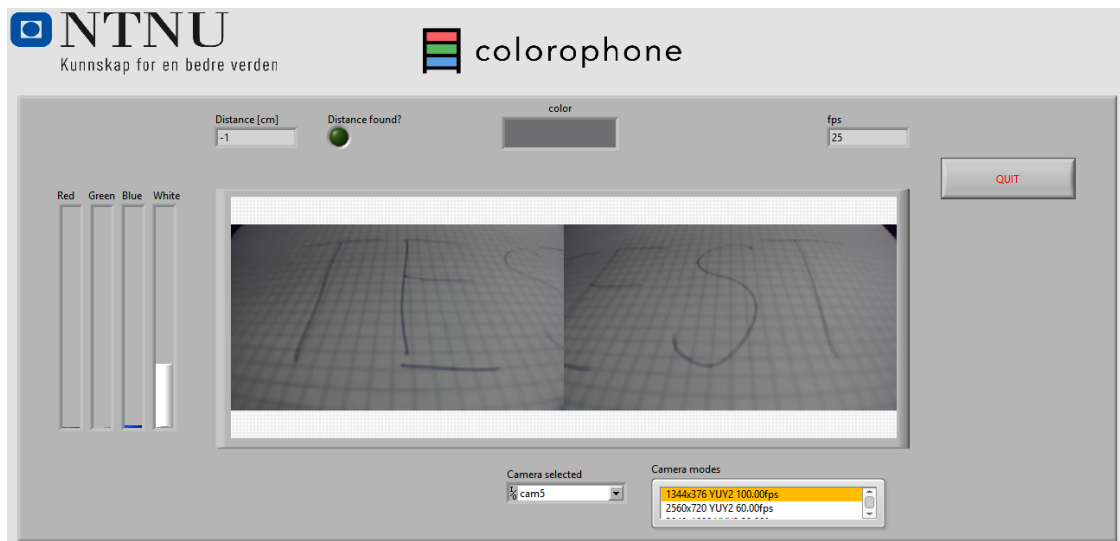


Figure 5.3.2: A picture showing what the front panel looks like to a regular user.

The developer mode has some additional GUI elements that aid in debugging the software, and giving the ability to change settings on the fly. There is also an extra tab for graphs that show data over time if the developer is interested in that. On the front panel the following additional elements are added for developer mode:

- **Four value indicators** for red, green, blue, and whiteness values, but as double type.
- **Disparity indicator** for showing the current disparity.
- **Clock FPS** that tells the number of times the clock triggers per second is shown

- **Sonification FPS** that tells how many times per second the software calculates sonification data.
- **Overview of the subprocess states** so that the developer can quickly see if a subprocess is in the wrong state based on the situation.
- **Four queue indicators** that keeps the developer updated on how many elements there are in each message queue for the subprocesses.
- **Error messages box** showing error messages.

The front panel during developer mode at run-time can be seen in Figure 5.3.3.

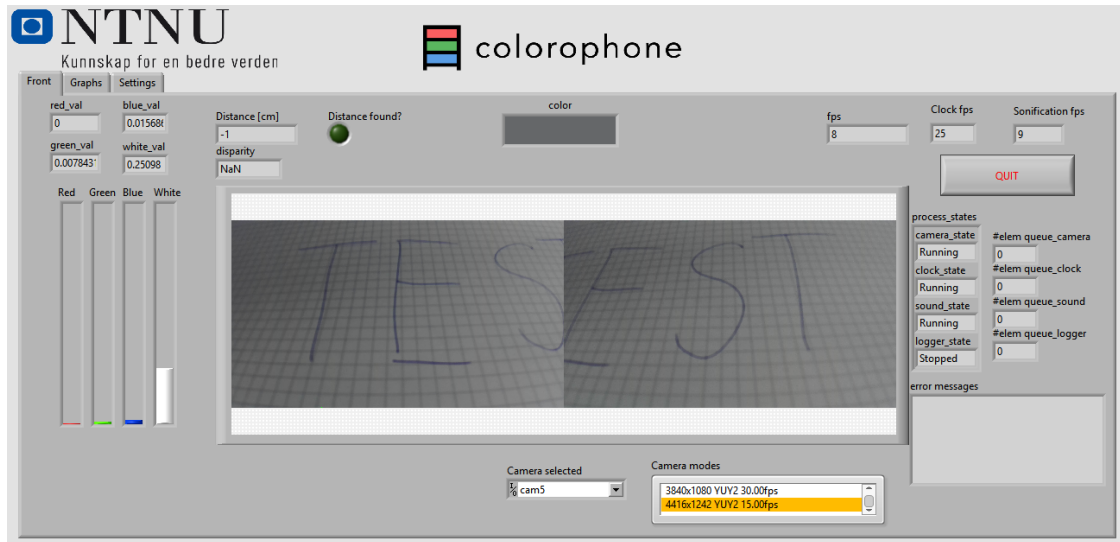


Figure 5.3.3: A picture showing what the front panel looks like when developer mode is enabled.

5.3.4 Application settings

Each subprocess has their own settings that can be changed in the 'Settings' tab that is available when 'developer_mode' is enabled. There are application based settings, and then there are the parameters. Parameters will be touched upon later. The application based settings are as follows:

- **dist_calc_method.** Sets the method that will be used for calculating the distance. By default set to "SSD" for the Sum of Squared Differences method. Has two other modes available; 'Random', and 'None'. 'Random' produces random distances, and 'None' sends out no distance.
- **log_path.** Defines where logs will be saved. By default set to "[PROJECT_DIR]\data\Log".
- **Logger Mode.** Turns logger on or off, is off by default.
- **desired_fps.** Determines how many times per second the *Clock* subprocess will send a 'Trigger' message to *GUI*, and thus determine the maximum number of camera frame captures per second for the application.

5.3.5 Parameters

Within the settings for certain subprocesses there are values that can be defined by the developer, and will be referred to as parameters in the rest of this thesis. What classifies as a parameter in our case, is whatever affects the resulting output of either the handling of the stereo camera frames, or the sonification based on the collected visual information from the aforementioned frames. This leaves us with the subprocesses *Camera* and *Sound* as subjects of interest when it comes to parameters.

Camera

The following parameters in the Camera subprocess all have significant effects on the resulting distance estimated by the distance calculation method:

- **filter_threshold:** DBL. Defines the threshold for counting similar disparity values. The most frequently occurring values, within the threshold, is passed through the filter.
- **minimum_similar_lines:** I32. Defines the minimum amount of similar disparity values, each corresponding to sets of lines in the current frame, that will give a trustworthy distance output.
- **length_ref_lines:** I32. Defines the length in pixels, for the reference lines in the left camera frame. The test lines in the right frame, are automatically adjusted to an appropriate length when editing this parameter.
- **height_area:** I32. Sets the height, in pixels, for the reference area, and test area.
- **number_of_lines:** I32. Sets the number of lines that are extracted from the camera frame with even spacing in between them.
- **calibration:** 2D array of DBL values. Calibration data used in performing interpolation between distance and disparity. The software also uses this calibration data to set the range of the distance function. More on the calibration procedure in Section 5.2.1.

For a look into how the distance calculation parameters were optimized for the prototype, see Section 6.1.1.

Sound

For the Sound subprocess there are numerous parameters that can be changed, as they all have a hand in making sure the sounds are pleasant to the ears of the user:

- **amp:** DBL value. Short for amplitude of a sound signal. Defines how loud a sound signal should be. All RGBW sound signals have a unique amp parameter referred to as amp_c . Here c represents one of the RGBW colors as R , G , B , or W . The distance sound signal has one as well, referred to as amp_D .

- **freq:** DBL. Short for the frequency of a sound signal, Referred to as $freq_c$. The red, green, and blue sound signals all have a unique frequency parameter. In our implementation, we utilized the same frequency values as the original Colorophone software, which were chosen based on studies done. These are frequencies that the average person finds the least annoying, but also frequencies that does not interfere with human speech. At the same time the frequencies have to differ so they are distinguishable. The red frequency is set to $1.7kHz$, green to $550Hz$, and blue to $150Hz$, meaning they are quite distinct and generally easy to identify after having done listening training.
- **freq_cutoff:** DBL. Exclusive to the white sound signal, and is used to cutoff higher frequencies in the signal. It was set to be $700Hz$ at default for our prototype.
- **min_freq** and **max_freq:** both DBL. Exclusive to the distance sound signal. Defines the frequency range of the distance sound signal.
- **min_distance** and **max_distance:** both DBL. Exclusive to the distance sound signal. Defines what distance range is acceptable for the distance sound signal.
- **volume:** DBL. Sets the base volume of the application. By default set to 1, meaning max volume. This is of little concern as the operating system volume is the one generally used to set the volume.

For a closer look into how the sonification parameters were optimized, see Section 6.1.2.

5.3.6 Distance calculation code

The distance calculation is performed in the **distance_ssd.vi** (see Appendix B.9 for code), which consists of subprocesses that performs all the tasks necessary to calculate the distance. This description only concerns the software implementation of the distance calculation, see Section 5.1 for a more conceptual description. The following describes all the succeeding steps, in the order they are preformed in the software:

- In **index_data_for_imaq_extract.vi** the pixel coordinates, that are needed to extract pixels from the camera frames, are generated through the use of the *Camera* parameters.
- In **IMAQ_extracting_horizontal_lines.vi** (see Appendix B.9 for code), pixels from the stereo camera frame is extracted, and will output the horizontal reference- and test lines. See Section 5.1.1 for more on what these lines are used for.
- The **ssd_method.vi** (see Appendix B.10 for code) implements the Sum.Sq.Diff. method to process each reference line with its corresponding test line. Thereafter it calculates the disparities for each set of lines, based on the placement of the valleys in the Sum.Sq.Diff. output arrays, and the array sizes.
- In **disparity_filtration.vi**, a band-pass filter first removes the disparities outside of a certain range (see Appendix B.11). Thereafter the .vi filters the disparities within range

with a mode based filter that passes through the most frequently occurring disparities within a defined threshold (see Appendix B.13). The disparity value for the current camera frame is finally found by taking the mean of the values passed through the filter. In addition to disparity filtering, this .vi has functionality for reliability control; it utilizes the "minimum_similar_lines" parameter to check if the amount of similar disparity values passed through the filter is high enough to produce a trustworthy distance output. If the outputs is regarded as trustworthy, a Boolean value is set true. This Boolean has influence on the "Distance found?" indication light in GUI.

- The **disparity_interpolation.vi** (see Appendix B.12) converts the final disparity value, for the current frame, to a distance value with the use of a look-up-table and interpolation. The look-up-table is generated trough calibration (see Section 5.2).

A full view of the entire procedure is shown in the Figure 5.3.4.

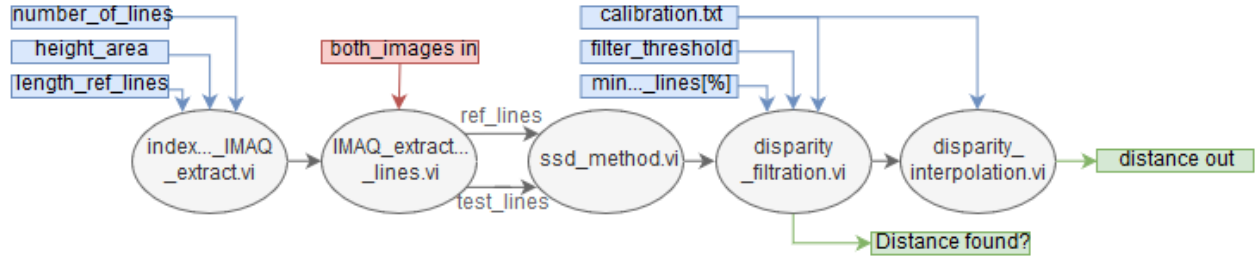


Figure 5.3.4: A simplified presentation of the distance calculation code architecture, with parameter and image inputs, and with outputs.

5.3.7 Method for color recognition

The system utilizes the color recognition code inspired by the original Colorophone system [1] in order to detect colors in a stereo camera frame. We were dealing with frames which contained both left and right images, so we decided to check the color for the left image only so that no fundamental changes had to be made on the original code.

What the method does is that it extracts the center row from the left image, and then the outer pixels of the row is removed, as we are only interested in a small area in the image center for this project.

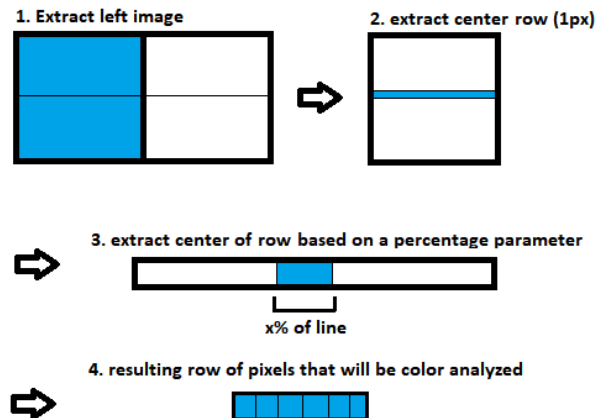


Figure 5.3.5: Illustration that clarifies how the pixel area extraction of the left image is performed in preparation of performing color recognition.

The amount of pixels remaining are based on a percentage constant that is set to 6.25%, eg. if the row is 500px wide, then the extracted area will only cover the center 31px of the row. See Figure 5.3.5 for a visual explanation on how this process is executed.

The newly extracted row of pixels is then fed into a function that calculates the mean RGB values across all the pixels. These mean values are then scaled based on a mathematical function that utilizes the two constants $EXP = 0.67$ and $k = 0.06527$. A whiteness value is calculated based on the resulting mean RGB values. The RGBW values are then passed on to the sonification process. For a closer look into how the code recognition works in LabVIEW, see Appendix B.4.

5.3.8 Sonification

The colors and the distance information needs to be transformed into sound for the user, and the name we use to describe the procedure is 'sonification'. The sonification procedure consists of two loops running in parallel within **sound.vi**, with the creation of sound signals happening in one loop, and the playing of these signals in the other loop. See Appendix B.5 and Appendix B.6 for a view of the code. It is important to note that the reader should have read about the *Sound* subprocess parameters for this software in Section 5.3.5 before continuing to read this subsection.

Color signals

To make explanations clearer; whenever 'color signal' is mentioned, we are referring to not only 'red', 'green', and 'blue' sound signals, but also the 'white' sound signal.

The **generate_color_signals.vi** generates an output waveform signal that consists of up to four added signals; red, green, blue, and white (see Appendix B.6 for code). The amplitude A_c for each signal is based on its corresponding RGBW intensity values $I_c = [0...1]$ retrieved from the color recognition process done in the *Camera* subprocess, and the corresponding color amplitude parameter amp_c . The actual amplitude for a color signal is determined by the formula shown in Equation 5.3.1. The division constant 255 is used so that amp_c can range from 0 to 255 without leading to signal clipping. So as an example, if no red color is detected in the defined focus area for the left image taken from a stereo camera frame, the intensity of the red sound signal will be set to zero, and thus it will not be heard. This applies to all of the RGBW signals.

$$A_c = I_c \frac{amp_c}{255} \quad (5.3.1)$$

The red, green, and blue signals are all pure sinetones, and their frequency is set based on their corresponding $freq_c$ parameter. The white signal is unique since it is represented

as white noise. The white noise fills the entire audio spectrum with frequencies, so a low-pass filter is utilized in order to remove frequencies higher than the defined frequency cutoff parameter $freq_{cutoff}$.

Tick signal

In **generate_tick_signal.vi** (see Appendix B.7 for code) a ramp wave is generated, where its frequency is based on the distance calculated by the distance calculation method used by *Camera*. See Section 5.1 for more on distance calculation method. By setting the ramp frequency f low enough, it will sound like a easily recognizable 'tick' sound.

The ramp frequency f is inversely calculated off of the distance from *Camera*, so a large distance value will create a ramp wave of low frequency, and a lower distance will cause the frequency to rise. The minimum- and maximum distance parameters decide what distances are acceptable in the creation of the ramp wave. If the distance is outside one of these limits, or is a negative value (indicating an error), then no ramp wave is generated. The minimum- and maximum frequency parameters are used in conjunction with the distance parameters to calculate the frequency slope, which is used in determining how many hertz are increased for every centimeter decrease in distance, and ultimately the ramp frequency f . At the same, these frequency parameters decide the lower- and upper limit for the frequency of the ramp wave.

5.3.9 Bugs

Unfortunately the software comes with its fair share of bugs, as it too is a prototype in and of itself. This section will elaborate bugs that will have a direct impact on the user utilizing the prototype software.

Tick spurs

If the user somehow ends up in a situation where the distance keeps jumping from valid to invalid, then the user will hear fast and spurious ticks. The cause for this is that the software has no way of handling a situation like that, so it simply starts by generating a ramp wave when receiving a valid distance, then it gets cut short when invalid distance is received. The problem is further illustrated in Figure 5.3.6.

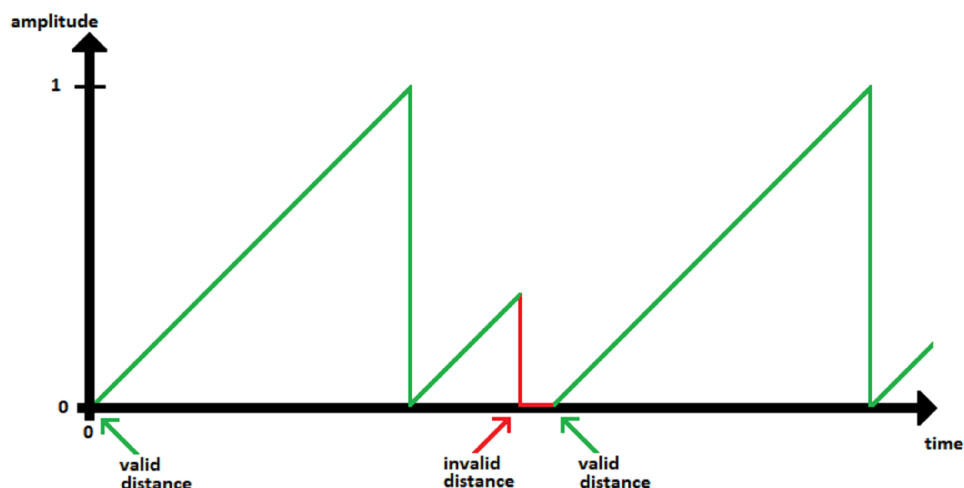


Figure 5.3.6: *An illustration to show what happens with the tick signal when the distance goes back and forth between valid and invalid. Red indicates a period where the distance is invalid, and green when its valid.*

There is another, more critical bug however. If the video mode or the selected camera is changed during run-time, then a lot of spurs will start occurring. If the software .exe was used, then the spurs will disappear if the application is reset, until video mode or selected camera is changed again. If the software is ran through the LabVIEW project, then the problem will persist until the project is restarted. We are not quite sure what the underlying cause is unfortunately.

5.4 3D design

Considering our background, it would seem appropriate to just design our prototype from the perspective from electrical engineers. However, our conquest is to see whether stereo cameras would benefit and progress the current colorophone design, as can be seen in Figure 1.2.1 in Section 1.2. By nature, all stereo cameras will be heavier and larger than the previous sensors used (one RGB camera and one ultrasonic sensor). Thereby it is important to our mission to test if we can successfully pack a stereo camera into a compact device that can be comfortably worn for a long period of time.

Because of the lightweight sensors previously used, the old design was made to fit the user like traditional eyeglasses. However, our design is bound to be heavier due to the new components, thereby from a design perspective it will look and feel like a hybrid mixture between eyeglasses, and augmented reality headsets.

The current design bears many conveniences that we aspire to keep, such as an elegant and modern design, and usage of open-air headphones. We will utilize the same headphones in

our design. The previous design also has a few drawbacks we seek to improve with the new design, as for example fixed side temples that cannot be adjusted to the user's head.

5.4.1 Open-ear headphones, AfterShokz

The headphones chosen was a pair of AfterShokz Sportz M2, which uses bone-conductive technology to deliver the audio to the user, and is known as open-ear headphones. The reasoning behind using open-ear headphones is that they will not occupy the users ears, and thus the user can more easily interpret environmental sounds, while also listening to audio delivered by the prototype.



Figure 5.4.1: Photo of the Aftershokz.¹

5.4.2 Implementing a 3D design from scratch

Creating a new headset is not a straight through process, as we are electrical engineers and not designers. Design is about construction and implementation. There are three limitations, imagination, production and materials.

To create our headset, we had to choose a 3D designing software. Thanks to the expansion of hobbyist 3D printing community, there has never been a time with as many options for 3D designing software. We decided to use Fusion 360, as it is free to use for students and non commercial use. Fusion 360 can generate material stress simulations, a function we will use to validate designs that we could not produce. The creators of Fusion 360 have developed

¹Photo taken from:
https://www.elfa.se/Web/WebShopImages/landscape_large/32/94/aftershokz-as321.jpg

many courses to teach the proper ways of utilizing their software, it took about a week to learn this software.

There are several production methodologies to transform the digital designs created with fusion 360 into physical parts, these methods have their pros and cons. During this project all the components we design will be created using a 3D printer, however, all our designs can be injection molded.

Injection molding is one of the cheapest and fastest production methods to mass produce a product, but there are some limitations. For each individual part there needs to be at least one unique mold, creating a mold is an expensive and time-consuming task. It is impossible to create one complete hollow part, if you need a hollow ball, then you must create two hollow half balls and then glue them together.

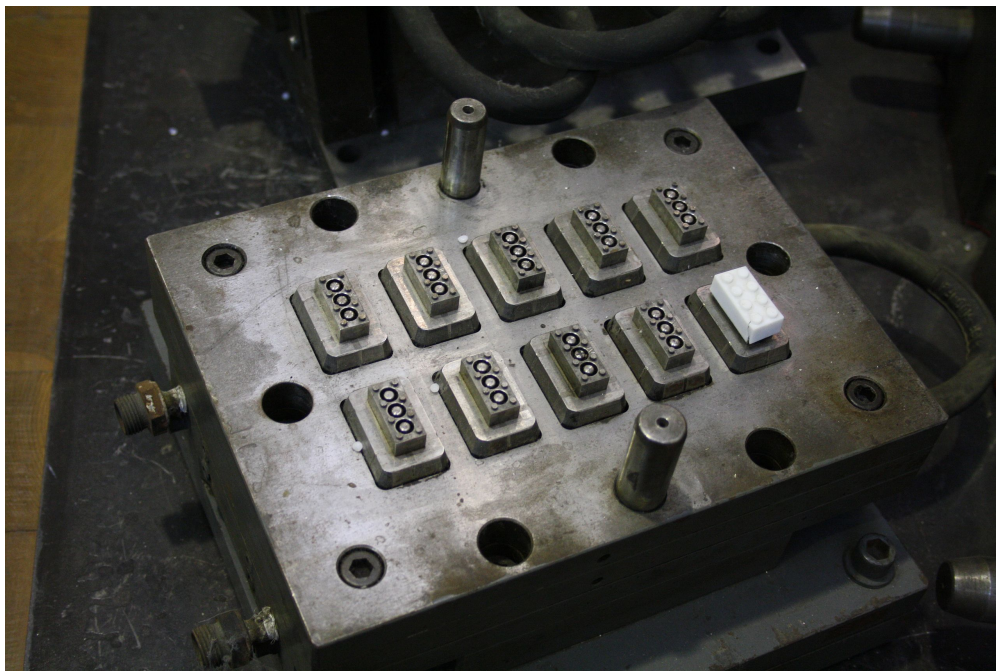


Figure 5.4.2: A Lego injection mold.²

The other production method is 3D printing, it allows for rapid and inexpensive prototyping. However, 3D printing sets some limitations on the design process. The digital design must have a large flat surface to be able to be repeatedly and successfully printed. This limits the aesthetic design and ergonomics, as most organic shapes have many curvatures. A 3D printed part made with one material, will always be weaker than the same design made with the same material using injection molding. This is due to the layer technique used to 3D print a part, as it negatively impacts the tensile strength. This phenomenon is described in further detail in Section 5.5.2. The price of 3D printing a single part costs less than

²Picture taken from:
https://en.wikipedia.org/wiki/Injection_moulding#/media/File:LegoSpritzguss.JPG

producing a single part with injection molding, but the unit price and production time will not be reduced when the production numbers are increased unlike injection molding.

Considering our time frame, and the fact that we are producing a prototype and not a final product, it makes perfect sense to utilize a 3D printer. This will limit our design, but we can at least be assured that a 3D printed component that is strong and durable, will only become better when injection molded. We will 3D print every design we make in PLA and TPU, but we will also do stress simulations as if the parts were made with injection molded PLA and ABS. More information about the materials can be found in Section 5.5.2.

Before we start the design process, it is important to note the desired and necessary attributes for the final product, and the limitations. So that we can keep them in mind while designing. The new design needs to have the following attributes and limitations.

- Elegant design that is understated and contemporary.
- Great ergonomics (the ability to conform to all head sizes and shapes).
- Lightweight construction, and good weight balance
- High strength construction that feels solid to the touch.
- High resistance against breakage and deformation caused by a fall or an impact.
- All surfaces that touch the skin must be malleable and comfortable.
- Each component must have at least one flat surface for it being able to be printed.
- The headset must contain the ZED Mini, and the Aftershokz headphones.
- The user should have clear eyesight with the prototype on.

Our 3D printer housing for the stereo camera will be consistent of 5 major components, and a total of 37 individual parts and screws. All the major components have several versions, we tried to see the good and bad from each design iteration, so that we could end up with a good solid design that satisfies our needs. During this chapter, we will go through the initial design process, as stage 1 gives great insight of how the design process progressed. But it will be incredibly time consuming for you the reader to read about every single version of each component. Hence, we have decided to proceed after stage 1 right up to the final stage. All major components will be displayed, and their functionalities will be explained.

But for those who are intrigued to know more about our design process, you can find more in our appendix, see Appendix C. You will find all the iterations from each component included with pictures, pros and cons. You will also find computer simulation for stress testing the parts, as if they had been injection molded with ABS, see Appendix C.1.

5.4.3 Stage 1 of the design process

The ZED Mini has a plastic shroud that protects the internals, this shroud also adds weight and bulk to the circuit board that contains the lenses and microchips. We will not be using

this shroud as we will be creating our housing that will protect the internals of the ZED Mini.



Figure 5.4.3: The ZED Mini internals.

As we can see from the figure 5.4.3, the circuit board is bolted to an aluminium frame. We reverse engineered this frame. We believe that the aluminium frame serves three purposes:

- The frame protects the circuit board from a high impact hit, because the frame is larger than the circuit board, assuring that an impact will always hit the frame before the circuit board.
- The frame reinforces the circuit board and adds torsional rigidity. Because it has many crossbeams and is tightly bolted to the circuit board. If the board shifts or bends, then the two camera lenses will not be in parallel to each other. Such a shift will negatively impact the accuracy for distance measuring at longer distances.
- The frame serves as a heatsink for the microchip onboard, as there is thermal grease placed on the touching surfaces between the frame and the microchip.

It is in our best interest to keep this aluminum frame and use it in our camera housing. It will be used as a structural part, leading to a lower weight penalty. The same number of hours that would have been spent on a new heatsink design can be allocated toward creating a more ergonomic and elegant design for the camera housing and temples.

The first digital part we designed was an exact replica of the ZED Mini internals, see Figure 5.4.4. All the important characteristics were measured with a measuring caliper. The idea is to use this digital model to help us design a better camera housing. As we can always be assured that the ZED Mini internals fit our camera housing without 3D printing the actual part.

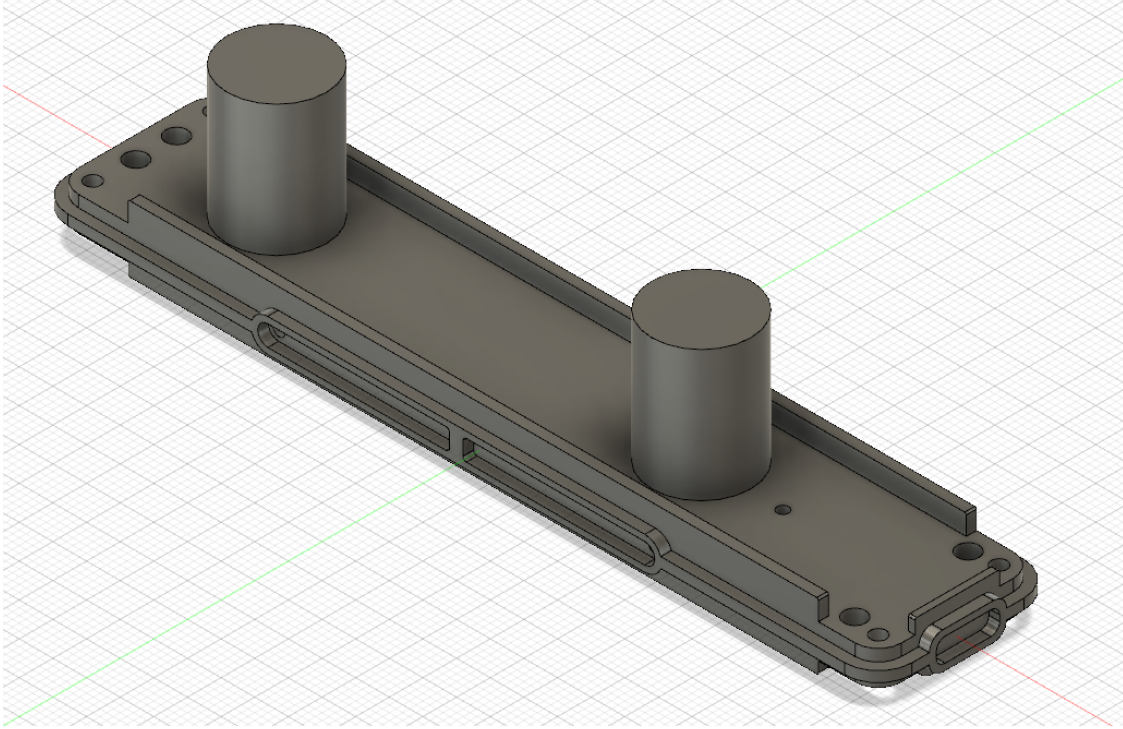


Figure 5.4.4: A 3D computer model replica of the ZED Mini internals.

A unique aspect of the ZED mini is that the lenses are not centered on the Circuit board. This meant that a design choice had to be made. Do we center the frame on the user's head (see left side of Figure 5.4.5), or, do we center the lenses on the user's head (see right side of Figure 5.4.5)? This sparked a discussion internally. Because some in group believe that if the lenses are not centered on the head, then the entire design will look wrong. This idea assumes that the pattern recognition part of our brain will associate the two camera lenses with eyes, and if they are not centered and symmetric, then we will immediately hate the design. The other part of the group believed that if the frame is not centered, then the 3D printed housing part must become larger, otherwise a part of the frame will protrude out of the side.

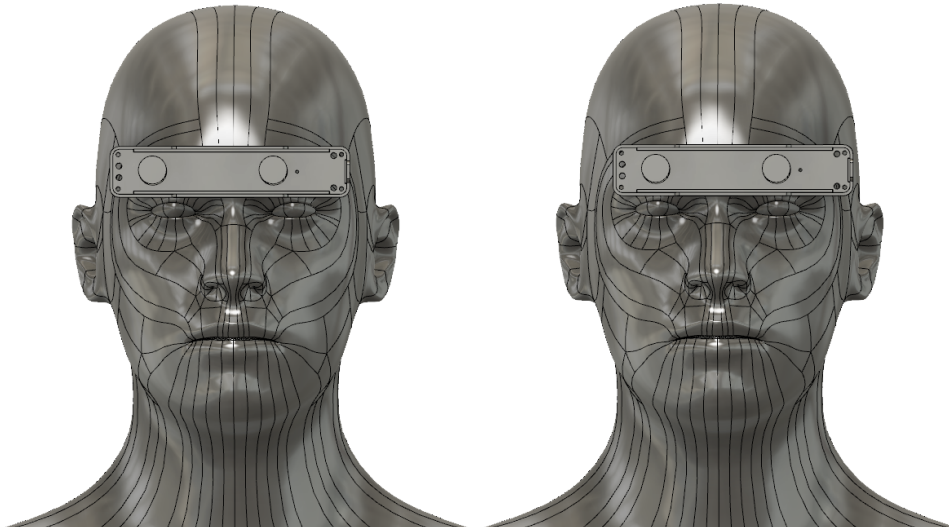


Figure 5.4.5: An illustration to show how centering the ZED Mini with the head, or lenses with the eyes looks like.

To end this discussion, two simple and wearable housings for the ZED Mini were made. These housing had two main objectives, display the importance of symmetry and give us a starting point for the next designs to come.

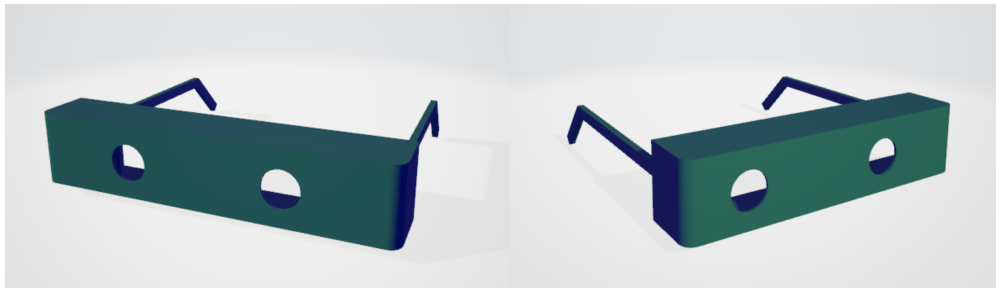


Figure 5.4.6: Left side is prototype V1 with centered lenses, right side is prototype V2 with centered aluminum frame.

Things we learned from the two 3D printed housing in Figure 5.4.6.

- The lenses must be centered on the user's head, almost every one that tried the housing on the right side on Figure 5.4.6 said it looked wrong. Mainly because of the asymmetric aesthetics.
- The extra width added to the housing on the left side on Figure 5.4.6 did add more weight to the unit, however the extra width made the prototype housing more comfortable to wear.
- It is nearly impossible to design a nose pad that will universally and comfortably fit every user. With just three subjects we found three different preferred shapes for the nose pad.

- Our suspension that fixed side temples (not user adjustable, do not confuse with collapsible) will be highly uncomfortable, was confirmed.
- The temples should be mounted on a collapsing mechanism, as a drop with fixed temples might possibly cause preventable damage.
- The design needs more curves, as it will make the housing more elegant and more comfortable to use.
- We need a secondary system besides the temples to help hold the housing on the user's head, as the temples alone are not enough to hold to housing during high speed maneuvers.

5.4.4 Final stage of 3D design



Figure 5.4.7: Final design of the housing

The housing was designed to conform around the user's head, as we can see in Figure 5.4.7. The front is tightly pulled over the ZED Mini, so that as little overhang over the user's face is achieved.

The backside is made larger than required for the camera frame, so that the frame wraps around the user's forehead, this choice was made for aesthetic and ergonomic reasons. The user can utilize the strap mechanism to tightly pull the housing on their forehead, and

the extra curvature of the backside helps spread the load over a larger surface area on the forehead. There is a backplate created with soft TPU that sits between the housing and the users head, see Figure 5.4.8.3. When the strap is used correctly, more weight will be placed on the ears rather than the nose, resulting a more comfortable fit.

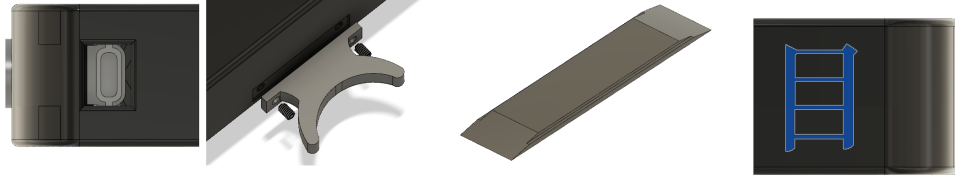


Figure 5.4.8: From left to right, 1. left side of the housing, 2. ingress in housing for nose pad and setscrew, 3. TPU backplate, 4. right side of the housing.

On the left side of the frame, as seen in Figure 5.4.8.1, we have an opening for the USB type-c cable for the ZED Mini. In the middle section there is an ingress and two holes, these are for the nose pad, as seen in Figure 5.4.8.2. The nose pad is not molded into the housing because we have several nose pad designs meant for different users. There are no extra strengthening measures taken in the center section, because we will be utilizing the aluminum frame from the ZED Mini. It will be tightly bolted to the two beams on the left and right side, see Figure 5.4.9.1. On the out right side there is a Colorophone logo ingress, this is to pay homage to the previous versions of the colorophone prototypes, the logo can be seen in Figure 5.4.8.4. A cross beam was placed on the inside of the right side, as a result of computer simulations revealing potential permanent damage during an impact after free-fall, see Figure 5.4.9.2.

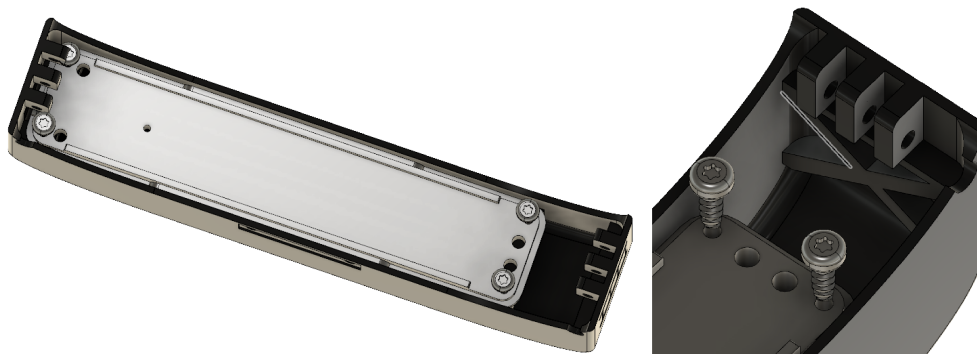


Figure 5.4.9: From left to right, 1. The light grey part is the aluminium ZED Mini frame inside the housing, 2. The crossbeam on the right side of the frame.

On each side of the frame we have the connecting points for the temples. Underneath the connection points we will find two holes, these serve two purposes. They allow a M3x20mm setscrews to enter the connection points so that the temples and frame can be screwed together, as seen in Figure 5.4.10.2. The connection points allow the temples to be folded during shipping, see Figure 5.4.10.1. When the user wants to deploy the temples, they can

just push the temples to the last position. Just before the temples hit the last position, the user will feel and hear a tactile click to confirm that the temples are fully deployed. This feature was achieved by letting the temples and frame slightly rub on each other right before the temples are in position. We observed no material erosion due to this mechanism, but the potential is there.

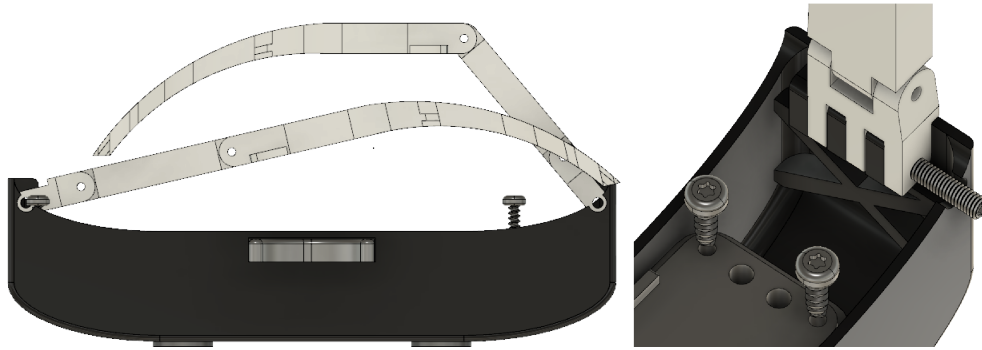


Figure 5.4.10: From left to right, 1. The housing with the temples folded, 2. The collapsing mechanism and the holes in the frame from the setscrews.

When purchasing traditional eyeglasses, one will find tens, if not hundreds of choices in just one store. We as users might initially differentiate the glasses based on visual appearance, but, different stores carry such a vast inventory simply due to the enormous pool of different head sizes and shapes. Designing one style of nose pad that can fit all users is unfortunately out of the picture, we decided on 3 different designs. The reason we stopped at four was simply time limitation, developing a library of different nose pads that could fit most users is definitely a feasible future.

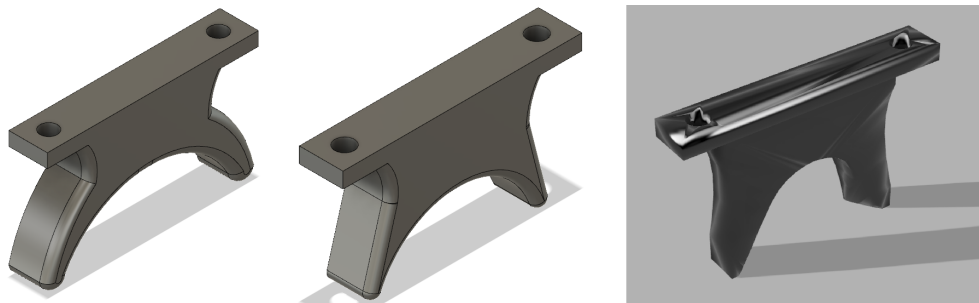


Figure 5.4.11: The three different nose pads that were designed.

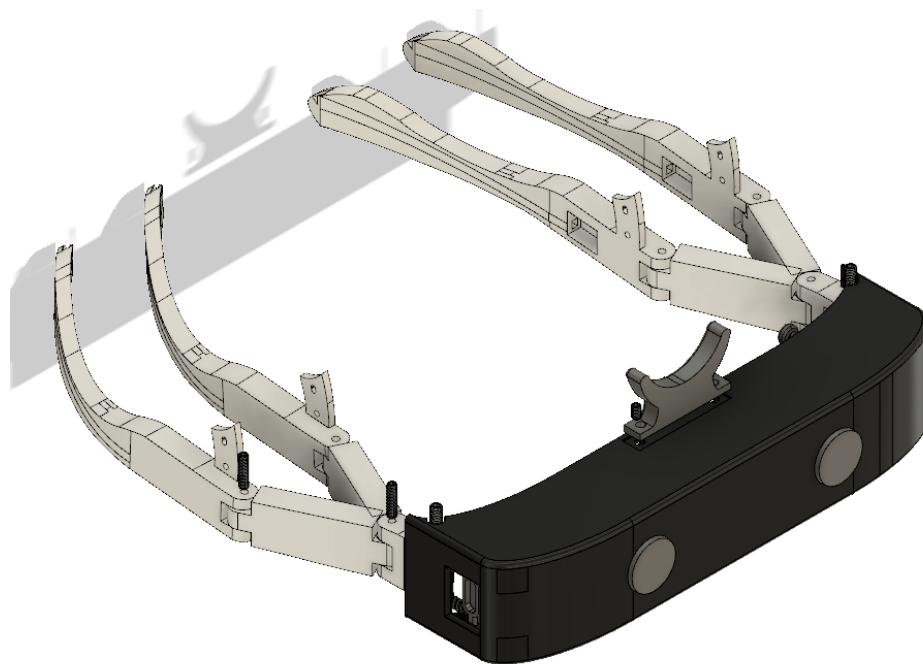


Figure 5.4.12: An illustration of the range of width for the temples.

The temples we have designed have several unique functions. The two joint mechanism allows the temples to have a range of width from 102mm to 166mm. This insures better compatibility with different head sizes and shapes. The temples have mounting points for the bone conducting headphones. The bone conducting headphones require some tension on the user's head to function properly. The temple adjustment allows for perfect tension on the headphones if they are adjusted correctly. Once the temples are adjusted, they will not hold position. Because the weight in the front of the housing will pull the temples apart with time, and high-speed movements. For optimal fit, the user or a helper should unscrew the setscrews that join the individual parts in the temples, not to be confused with the setscrews that join the temples to the housing. Once unscrewed, the user or helper should apply Loctite to the setscrews. The setscrews should be tightened to the temples promptly, then the user should wear the head device and form it to their head. In a couple of minutes, the Loctite will harden, and the temples will keep their new shape. the last section of the temples is hollow, so that a wire or string can be threaded through. The string is used as a strap.

Because the temples are produced with hard plastic, we have created thin TPU side-plates that will be fused on the inside of the temples, as can be seen in Figure 5.4.13. These side plates add more comfort because of the malleable nature of. They were also designed to hide the headphones wires, and the strap mounting point. During the Loctite applying procedure the user might have trouble aligning the parts of the temples, but with the side plate on, the components will be held together.



Figure 5.4.13: The TPU sideplate that are fused to the temples

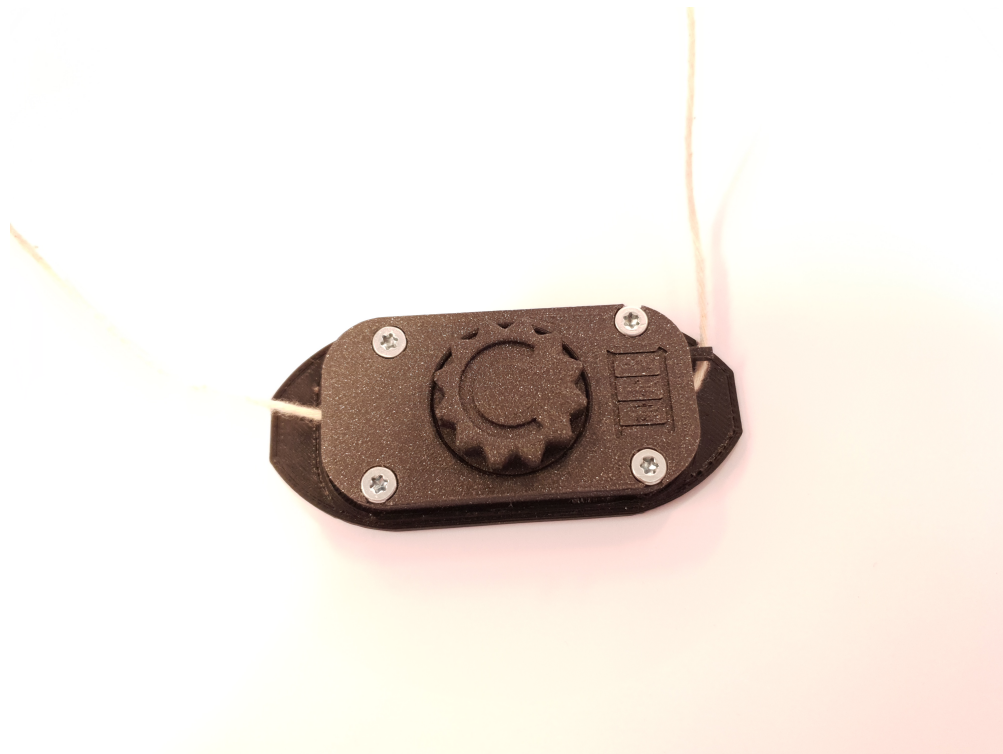


Figure 5.4.14: The strap mechanism for the prototype.

The strap tensioning mechanism is incredibly easy to use. During production, a wire must be threaded through the tensions, and then mounted to the temples. When the user wears the head device, they can utilize this mechanism for better fitment. The geared wheel, as seen in Figure 5.4.14, can be turned clockwise to tighten the strap. The geared wheel is designed to maintain its position once tightened. The only way to relieve the tension is by pulling down on the lever on the right side, and then rotating the geared wheel in an anti-clockwise direction.

5.5 3D printing

5.5.1 3D printer, Wanhao Duplicator i3 Plus

The 3D printed parts designed for our prototype were printed with a Wanhao Duplicator i3 Plus. The printer in question was a FFF style printer, which prints the 3D object layer by layer, and had to be taken into consideration when designing the prototypes [24]. The 3D printer had been modified, and was being maintained by Jon Petter. The modifications had a significant improvement on the quality of the printed pieces, and were as follows:

- **Micro Swiss All Metal Hot End.** Replacement for the stock nozzle and thermal barrier tube, enabling the printer to print with higher temperatures, higher speeds, and composite materials due to a significantly better heat transfer. In addition, it gave a more consistent flow and melting rate, while also reducing stringing.
- **Aluminum Y Carriage Plate.** A much sturdier and lighter Y carriage plate was added since the stock Y carriage plate was too weak, and prone to bending, which affected the prints by messing up the build plate calibration.
- **Better Calibration Thumbwheels.** Bigger thumbwheels with better grip and dial, which made the calibration process much easier as one could see how much each thumbwheel had shifted since the last calibration. It was also now capable of using a locknut for better fastening of the thumbwheels if needed.
- **Stiffer Bed Plate Springs.** Stiffer bed plate springs together with spring cups helped to reduce wobble caused by the bed plate moving back and forward during printing, and worked as good support in keeping the bed plate calibrated. If just one or more of the springs are weakened, it is easier for the bed to wobble and mess up the calibration.
- **Z-brace Mod.** A highly recommended modification for the i3 styled printers, whereby adding a brace to the tower (z-axis) stiffens the chassis of the 3D printer, and thus negates tower wobble.
- **Glass Print Bed.** A removable glass bed for printing on that was placed directly on top of the original build plate. It could be removed from the printer quite easily in order to remove prints faster. At the same time it kept the process cleaner, and since it worked as a form for adhesion, printed parts would end up with a smoother bottom surface.
- **OctoPrint with Raspberry Pi.** A Raspberry Pi that is connected to the printer through custom OctoPrint software³, adding a web interface for the 3D printer, and giving the user full wireless control of the printer, with built in webcam support and GCODE visualizer. This mod did not directly affect the print quality, but gave much better control and surveillance of the printer.
- **PID tuning and various other software settings.** PID-tuning was ran after the Micro Swiss upgrade was completed, which caused the temperatures to be much more stable during printing, while at the same time never dipping more than 2 degrees

centigrade from the target temperature. Other settings such as jerk, and acceleration was modified to give a much better print quality overall as well.

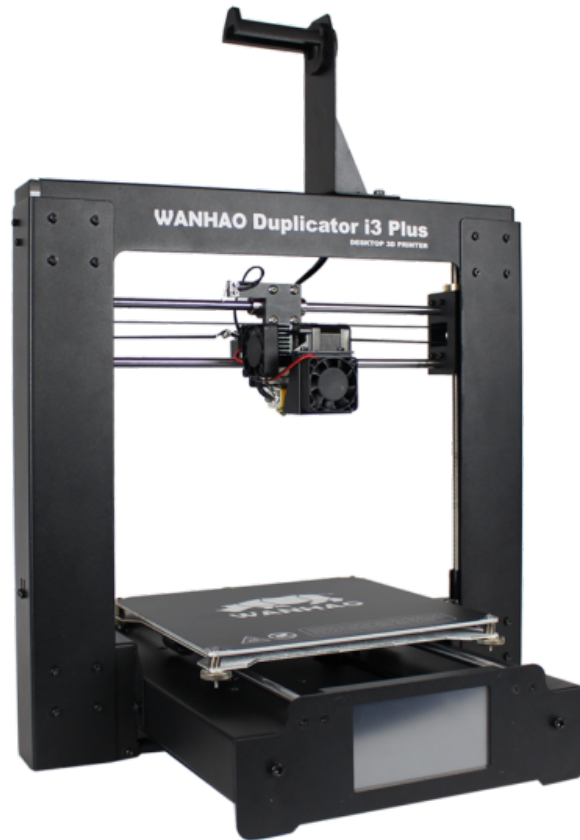


Figure 5.5.1: A stock Wanhao Duplicator i3 Plus⁴

5.5.2 3D printing process

During the 3D design process for the prototype, several prototype versions were being printed so that their strengths and weaknesses would arise. It became apparent after some stern stress testing that the printing orientation had a noticeable effect on the prints, as the strength between the layers (z-axis) was weaker than any other axis (see Figure 5.5.2).

Another factor is the type of plastic that is used, as most 3D printers are capable of printing with different types of plastics and composites, all ranging from standard PLA plastic to

³<https://octoprint.org/>

⁴Picture taken from:
https://cdn4.mystore4.no/thumb/480_600/3dprinter/66705_Wanhao_Duplicator_i3_Plus_1.png

⁵Picture taken from:
<https://s3-eu-west-1.amazonaws.com/3dhubs-knowledgebase/print-orientation/visual2.png>

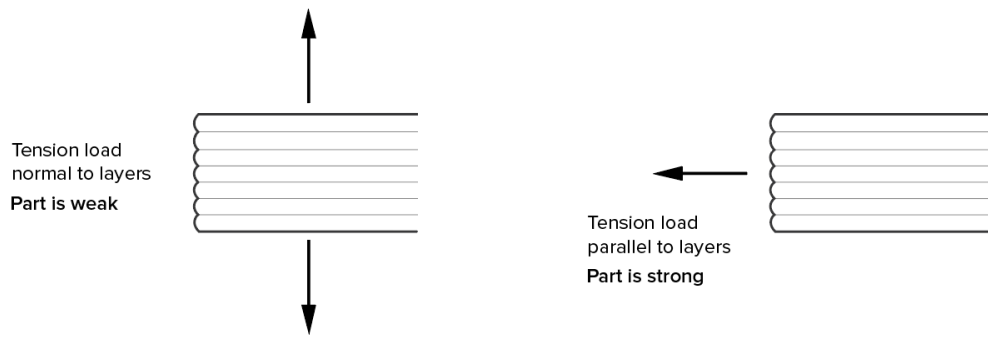


Figure 5.5.2: Illustration of the directional tensile strength for 3D printed parts⁵.

different types of plastic composites, from wood to carbon fibre [25]. The materials have different physical properties and have their own advantages and disadvantages, as some of the materials are optimal for mechanical parts, while others materials only works best for decorations.

The early prototypes has been printed with PLA as it is easy to work with and easily available. A few different types of PLA have been used. At first, *Flashforge* PLA bought from the local *Clas Ohlson* were used⁶, but due to several failed prints using the PLA from *Flashforge*, it was switched to PLA filament from *Prusa Research*⁷ (see Figure 5.5.3), as their filament are made with much higher standards than the rest of the industry.



Figure 5.5.3: The prototype stereo camera housing in the midst of being printed with black PLA from *Prusa Research*.

⁶<https://www.clasohlson.com/no/Filament-PLA-til-3D-skriver-Flashforge/38-7721-2>

⁷<https://shop.prusa3d.com/en/prusament/711-prusament-pla-prusa-galaxy-black-1kg.html>

Other types of filaments have also been considered, as PLA is not the best material for the finished product. PLA has a low glass transition temperature at 60 °C (The temperature of which the material begins to become soft) and has a low UV-resistance, which is not suitable for outdoor use. ABS was one of the considered materials as it is a strong material and can withstand high temperatures. However, the ABS requires high temperatures to print with to prevent warping and destroying the print. It also produces an unpleasant odor when printing, and from own experiences, not worth the effort to print with.

PETG from *3DNet*⁸ was later selected as a candidate, as it is a tough material that is very durable, can be used outdoors due to its higher UV-resistance, and is easier to print than ABS and without the odor. However, some problems still occurred when printing with PETG, as it has a tendency to not bridge well and support material for overhangs had a tendency to fail, making overhangs difficult to print if support is needed. This led to the choice of not using PETG during prototyping, as it is easier to get better quality prints from PLA for prototyping, and the color we received from ordering were blue, not black as requested.

The last type of plastic that has been considered is TPU. There are several types of TPU, but ended up choosing *Ninatek's Ninjaflex*⁹, which is a flexible material, but is at the same time relatively easy to print with some practice, as it a flexible filament, some calibration of the print speed and material flow is required to achieve a steady flow of plastic. Printing this material with the printer mentioned in Section 5.5.1 will be slow as the only way to print with this material successfully is to print with slow speeds to keep an even material flow. However, it is possible to purchase an upgrade for the printer called a *Flexion Extruder*¹⁰ that is specifically made to be used when printing flexible materials, but due to the high price and little usage of the flexible material it is not worth it. The plastic is too flexible to use for the housing for the camera, but is intended to be used as a padding material to give the user a more comfortable fit.

⁸<https://3dnet.no/products/petg-1-75-1-0-kg>

⁹<https://3dnet.no/products/ninjaflex-1-75>

¹⁰<https://flexionextruder.com/>

Chapter 6: The prototype

The final prototype seen in Figure 6.0.1 consists of the 3D printed housing (see Section 5.5) containing the ZED Mini, and bone-conductive earbuds (see Section 5.4.1) for sound. The prototype must be connected to a computer running the software for it to work.

The prototype had to be tested for us to be able to conclude just how well it worked in practice. In addition it was of great interest to see if a test subject could learn to make use of the prototype after having done specialized training exercises developed for the prototype, and then utilize the training to tackle problems that would require that the test subject could decode the sound signals into usable information about color and distance.



Figure 6.0.1: *What the final prototype looks like.*

6.1 Optimizing

Before doing any testing of the prototype, it first a necessity to optimize the software as best we could through proper parameters values which had been chosen based on intuitive reasoning. See Section 5.3.5 for a presentation on the software parameters and their effects in general. This subsection will first elaborate on the chosen parameters for the distance calculations, and then parameters for the sonification process will be justified.

6.1.1 Distance calculation parameters

We will now explain why and how the particular parameter values utilized by the distance calculation method were chosen.

From the conclusions made in Section 5.1.4, it was recommended to choose a threshold slightly higher than the minimum value 5. If we then choose 10, we must also divide it by a ratio value, since the prototype utilizes a resolution of 1344×376 , whereas the filtering example used 4416×1242 . We can find this ratio by looking at the disparity values at distance 30cm in Figure 5.2.3 and Figure 5.2.2 in Section 5.2.2, then find the ratio between those disparities. The ratio is then $\frac{300}{77} \approx 3.896$, giving us a threshold of $\frac{10}{3.896} \approx 3$ rounded to the nearest. Therefore `filter_threshold` was set to 3.

The `minimum_similar_lines` parameter was set to 15% of the total number of lines calculated. In Figure 5.1.11 under Section 5.1.4, the number of lines passed through the filter is 44 of 160 total, so 27,5% is passed through. Based on this it is estimated that a minimum number of similar lines less than 15% is unreliable.

The `length_ref_lines` parameter was set to 150px . When setting the length of the reference lines, it is important to notice that all the objects contained within the reference area is targeted by the distance calculation. See Section 5.1.1 for more on distance input data. Some length is needed in order to provide enough pixel data to make the distance calculation possible. On the other hand, multiple objects with different distances can result in unstable results, so 150px was chosen as an appropriate compromise.

The `height_area` parameter was set to 30px . The distance function performs best if the targeted object roughly covers the reference area in height. Therefore the minimum size of the objects that the distance function should be able to detect, is a key factor when deciding the height of the reference area. Within the defined range, 30px is enough to capture the height of a human head at roughly 4m .

The `number_of_lines` parameter is preferred to be as high as possible when regarding the reliability of the distance measurement, and since the resolution is set to a minimum, the parameter's effect on the real-time-performance is minimal. However, it had to be limited to 30, because there can not be more lines than the size of `height_area`.

The calibration parameter is set to the calibration data seen in Figure 5.2.3 found in Section 5.2.2. Since the sensitivity is so low at high distances, we only use calibration data up to 300cm , because anything further than that starts to become unstable. Thus we utilize calibration data ranging from 30cm to 300cm .

Figure 6.1.1 illustrates what areas will be used to calculate the distance based on the parameter values that we just defined, and are shown in black. The implementation of the distance calculation is elaborated on in Section 5.1.



Figure 6.1.1: *Black boxes shown indicate what pixel data is being used to estimate the distance.*

6.1.2 Sonification parameters

In the original Colorophone software the parameter frequencies chosen for the RGB sound signals were supported by studies, but their corresponding amplitude parameters were not, which meant that finding proper values for the amplitude parameters were a reasonable action to take. We thought it would make the most sense if each signal sounded equally loud when their corresponding color values were at full intensity, so we had to determine what the values for the amplitude parameters needed to be. For this we had to rely on previous data on how humans perceive loudness based on frequency, and we had to look no further than the ISO 226 standard¹, which consists of an equal-loudness contour that indicates how loud a sine tone of a given frequency is perceived by the average person. Through the use of the inverse ISO 266 plot seen in Figure 6.1.2, we found the values that would make the color sound signals sound equally loud at full intensity. The rest of this subsection relies on the fact that the user read about the Sound subprocess parameters in Section 5.3.5 and understands their purpose.

First, we assumed $freq_R = 1.7kHz$ was roughly $0dB \approx 1$ in magnitude, as can be seen in Figure 6.1.2, so we take the reciprocal of that and get the amplitude $amp_R = \frac{1}{1} = 1$ for red. We repeated this process for green and blue as well, still going by the inverse ISO 226 plot in the figure. For green we found that $freq_G = 550Hz$ was estimated to be at around $-1.5dB \approx 0.8414$, meaning $amp_G = \frac{1}{0.8414} \approx 1.19$. Further on with blue where $freq_B = 150Hz$, a magnitude of $-14.5dB \approx 0.1884$, and thus $amp_B = \frac{1}{0.1884} \approx 5.26$.

There was however uncertainty on how to deal with the white sound signal, since the ISO 226 standard is based on sine tones alone, meaning white noise would not fit the model. In the end, it was decided that it would probably be for the best to pick a value that sounded roughly the same as the primary color sound signals, at the cost of it being a subjective choice. It was agreed upon that $amp_W = 4$ was an alright selection for the whiteness.

¹ISO 226: <https://www.iso.org/standard/34222.html>

²Picture taken from:

<https://upload.wikimedia.org/wikipedia/en/thumb/c/c2/Lindos3.svg/800px-Lindos3.svg.png>

Lastly there were the sonification parameters for anything related to distance, which was the amplitude for the tick sound, and its minimum and maximum frequencies. This could also not be based on the ISO 226 standard, as it was a ramp wave, and filtering it was out of the question as it would change its sonic qualities so that it would no longer sound like a tick. Therefore the tick received the same treatment as the white sound signal, and was given a amplitude of $amp_d = 2.5$. When it came to the frequency range, the minimum was selected to be $min_freq = 1Hz$, as anything lower would have more than one second delay between each tick. The reasoning for this choice, is that too much delay between each tick will make it difficult for the user to know if a distance was found or not, since the software generates no tick if the distance value is invalid. The highest frequency was chosen to be $max_freq = 6Hz$, which we felt was justified as any higher frequencies made the tick sound more like a continuous tone, and less like a tick.

There is also a need to define the minimum- and maximum distance, which we base on the calibration data in Section 6.1.1, so only a distance in the range of $30cm$ and $300cm$ will generate a tick sound.

6.2 Testing

As with every prototype using sensors, there needs to be done tests or experiments were performance, reliability, accuracy, and other sensory qualities are under scrutiny. Thus we planned and executed tests for the following concepts; color recognition, distance recognition, and a combination of both color and distance. These tests will be referred to as 'the color test', 'the distance test', and 'the combined test'.

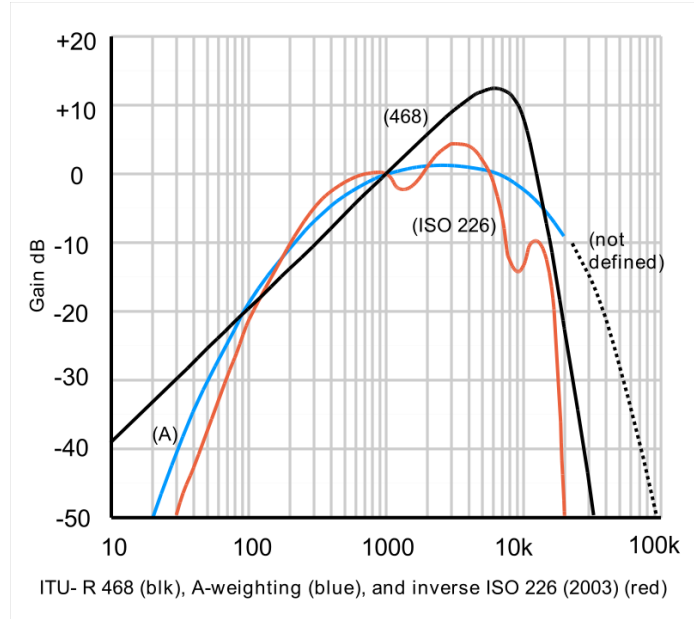


Figure 6.1.2: A plot of different weighing functions used based on the ISO 226 standard. In our case, what was of interest in this graph was the inverse ISO 226 plot in red.²

Before each test, the subject was trained in analyzing the sound signals provided by the software sonification procedure, such as relating sound signals to colors, or differing distance values in centimetres. The combined test would then be the final test, where the subject would need to utilize his training and experiences in conjunction with the prototype, so that he could decode the provided sound into visual information about the environment to solve a problem.

The training and testing of the selected subject was all done in a relatively quiet and uncontested, medium sized student work room at NTNU. All of the three prototype tests with a subject were carried out in May 2019, more specifically, color recognition on the 21st, distance recognition on the 23rd, and finally combined on the 24th.

When training and doing tests, the idea was to remove all variables that would complicate the mental tasks the subject would have to do during the experiments. We therefore decided that using a laptop to run the software was an alright decision, as it should not have had a particular impact on the prototype tests.

Regular consumer earbuds were used in place of the bone-conductive earbuds during the training and testing. The reasoning for utilizing these consumer earbuds was that they had longer cables than the bone-conductive ones, which lead to a less restrictive environment for the subject since a laptop was being used for tests. We believed this was fine, as the main goal of the color and distance tests were to see if the subject could mentally decode the generated sounds into useful information.

One of the project members, Jørgen, was the trainer, whereas another member, Magnus, was the subject. The subject was not visually impaired except for partial color blindness, but the training and testing processes elaborated on in this section can still be used on a visually impaired person. Of all the members, the subject was chosen because he had personal experience with music, and thus had a good set of ears. We thought a subject with good ears would more closely match the well developed hearing of a visually impaired individual, as it becomes second nature for such an individual to rely on their hearing to aid in understanding and navigating their immediate environment.

To train the subject with controlled inputs through the LabVIEW application, one has to set the 'training_sonification' conditional disable symbol in the software project to 'true', which will then let the trainer manually input what values he wants for red, green, blue, white, and the distance. White is technically calculated based on the RGB values during normal operation, but we added the option to set it manually so that the trainer can more easily decide what sound signals will be generated by the software.

6.2.1 Training color recognition

This training exercise introduces the subject to the unique sound signals that are used to represent red, green, blue, and white. It also lets the subject experience combinations of these unique signals, as it is crucial that the subject learns to identify each of the four components when they are playing at the same time. The subject is gone through the exercise in such a way that the association he has for each unique signal and its conceptual color is mentally solidified. This is done with the intent of granting the subject the ability to guess what colors the sound represents, similar somewhat to how individuals with synesthesia perceives color through sound. All that is needed to do this exercise a computer running the software, and earbuds to play the application sounds for the subject. The tick signal is turned off in the application settings so that only the color signals can be heard.

The color training exercise in steps

The exercise consists of the following steps:

1. In the first step of the color training process, the subject will get used to the four unique color signal sounds. To do this, the trainer begins by introducing the subject to the sound of all the three basic color signals at full intensity, including the white noise which indicates whiteness. Subject is then told that he can ask for a single sound, red, green, blue, or white, and the trainer will activate that sound manually. The subject is for the rest of this step given free reign to listen to each of the four signals until he feels confident enough in that he can recognize any of them when heard.
2. A small test is then done on the subject, so as to further solidify the connections that the subject has with the four signals, and their corresponding colors. The trainer picks at random one of the four color signals at full intensity, and then the subject guesses what color the signal represents. The trainer then replies what the actual color was, and notes down whether the subject was correct or not. This is done **twenty times** to make sure that all four unique signals are checked multiple times with the subject. If desired, this step may be repeated as many times as the subject agrees to, and then the accuracy for each execution may be compared to see how the subject has improved over time, if needed.
3. Moving on to the third step, combinations of two color signals are exposed to the subject, but is restricted to red, green, and blue signals, since whiteness is in reality derived from those three colors, meaning it also indicates how desaturated the color is. Desaturated color training is not included in this training exercise. The color combinations are referred to as the secondary colors. The secondaries are shown in Table 6.2.1. As with step one, the trainer introduces the sound of the secondary color signals, informs the subject what colors the signal is composed of, and what the resulting secondary color signal is called based on the table. The subject then familiarizes himself with the secondary color signals until he is ready for the next step.

4. Same as step two, but with the secondary color signals instead of the four unique color signals. This time however, only **fifteen** random secondary color iterations are executed for the subject. The step may be repeated however many times if wanted.

Table 6.2.1: How the three secondary colors are represented through combinations of RGB values.

Color combination	Resulting color
Red (255) + Green (255)	Yellow (255,255,0)
Green (255) + Blue (255)	Cyan (0,255,255)
Red (255) + Blue (255)	Purple (255,0,255)

Results

The time spent, and observations made for each step is summarized in the following list:

1. Took five minutes. Subjected tried to replicate the sounds vocally in order to aid in memorization.
2. Took ten minutes. The subject guessed correctly on all twenty samples, therefore giving a accuracy of 100%.
3. Took five minutes. Subject focused on identifying what signals the combined signals were made of.
4. Took roughly twenty minutes, since the subject spent more time listening to the combined signals, and the trainer needed more time to properly change to a new signal. The subjected managed to guess right on fourteen of the fifteen samples, leaving him at an accuracy of 93.3%.

After completing the exercise, and then calculating the accuracy results, it was decided that the subject was now ready for the color recognition testing.

6.2.2 Testing color recognition

In the same vein as in the original Colorophone paper, colored yogurt containers were used as objects for the subject to examine. The task for the subject was to correctly guess the color for every container that was examined with the prototype, which meant through sound only, and no visual information. It is very important to note however that the results from this test will be unreliable if the subject knows about the colors of the containers beforehand through experience or having been told.

There were five different types of yogurts of different colors, and they can be seen in Figure 6.2.1 as beige, red, green, yellow, and blue. With these colored containers we were able to provide sound signals that consisted of interesting combinations of red, green, blue, including the calculated whiteness. The RGBW colors calculated by the software of each container was noted down, and can be seen in Table 6.2.2. Do note however that the lighting in the room possibly had a big impact on how the stereo camera perceived the colors.



Figure 6.2.1: Yogurt containers used in the color recognition testing.
Top: beige, red, green. Bottom: yellow, blue.

Table 6.2.2: RGBW values calculated by the prototype software when the ZED Mini is looking directly at one of the yogurt containers shown in Figure 6.2.1.

Container color	RGBW values from software
Beige	(20,20,0,104)
Red	(158,0,18,0)
Green	(0,115,51,0)
Yellow	(77,102,0,0)
Blue	(20,0,140,0)

Testing preparation

Subject wears the prototype connected to a computer so that it rests above the nose ridge, then tightens the prototype so that it comfortably exerts force towards the frontal bone of the skull. Earbuds of appropriate length connected to the computer are placed in the ears. If the subject is not visually impaired, cover the eyes so that the subject can not see, which we did for our testing as shown in Figure 6.2.2. The tick sound is turned off in application settings, as they are of no interest in this particular test.



Figure 6.2.2: The subject prepared and ready for color recognition testing. The hood is acting as a blindfold.

How to conduct the test

Trainer randomly selects one of the five yogurt containers, and then gives it to the subject. The subject then examines the container with the prototype and reply what color they think the container is. This is performed twenty-five times to make sure all the different containers are examined multiple times, with the real color and subject reply written down by the trainer for each sample. Since the subject is assumed to not be specifically trained or expected to use perfectly correct color lingo for describing colors, combinations of simple colors may be used like 'slight yellow with strong white' for the beige yogurt. It is the trainers responsibility to decide if the replies used to describe the color correctly matches the color of the container. An accuracy percentage is then calculated to check how successful the subject was overall at individually identifying the colors of the containers.

Results

It has to be specified that the trainer in our case had to hold the container in front of the prototype so that it could be properly examined, since there were issues with the reflected lighting coming from the outside, which made it hard for the subject to know where to look on the container without focusing too much on the highlights. This was considered to be an alright exception to the test, as we would later in the combined test check how the subject managed to utilize the prototype on his own without human interference.

The test took thirty minutes, and the results were not quite as expected. Of the all the twenty-five samples, only eight of the replies given by the subject were considered to be correct, giving an accuracy of 32%. It was soon realized that a critical mistake had been made. We had assumed that the subject did not know about the yogurt container colors

beforehand, which was clearly false as he had seen them before. The results were thus not satisfactory, and ended up being unreliable when gauging the subjects ability to decode the color signals provided by the prototype. Due to this fluke in procedure, it was decided that the test would be done one more time after the subject had done a improvised training session with the containers, with the assumption that he knew only one of those five containers would appear in the test. In this improvisation the subject learned to recognize the five containers and their corresponding color name in a span of thirty minutes, meaning there would be a one in five chance of him guessing a container correctly during the second test run.

The second test took twenty minutes, and had pleasing results. Out of the twenty-five randomly chosen samples, the subject guessed twenty-three samples correctly, an accuracy of 92%. There was a clear improvement in performance, which was hypothesised to not only be caused by the improvised training, but also because the list of possible replies had been reduced to only five choices.

Although we can not make a perfect case for the first color test, it can be concluded that there was a clear improvement in performance when the subject was trained to recognize the sound of an object that had been taken from a group of known, and physically similar objects that wore unique, and recognizable base colors.

6.2.3 Training distance recognition

The point of this training exercise is to introduce the subject to the tick sound signal, and aid him in mentally connecting actual distances to the frequency of the tick sound heard. The idea is that the subject should be able to somewhat gauge the distance to whatever object he is looking at, and then create a mental map in his head. All that is needed for this exercise is a computer running the software, and earbuds to send sounds to the subject. Color signals are turned off in the application settings so that only the ticks can be heard.

The distance training exercise in steps

The exercise consists of the following steps:

1. The trainer lets the subject hear what the tick sounds like at minimum- and maximum distance, and notifies what distance is being represented. The trainer then linearly increments the distance from the minimum to the maximum acceptable distance value by a value X in centimeters as defined by the trainer, so that the subject can get used to how the frequency changes with distance. The trainer must remember to always tell the subject what the new distance is so the subject can memorize it. The subject is then free to ask for any distances, and listen to the resulting ticks until he feels confident in that he can mentally decode tick frequency to distance information in centimeters.
2. This step is done to solidify what the subject has learned in the previous step. The trainer randomly selects a distance value that resides within the minimum- and maxi-

mum distances, where it needs to be a number rounded to the nearest ten. The subject is placed in such a way that he under no circumstances will be able to deduce what distance values the trainer enters into the software. The subject guesses a distance value in centimeters, and the trainer notes down the response as well as the actual distance value. Do twenty of these selections. When complete, for all twenty samples, calculate the subject's accuracy, where any guess within a certain threshold of the actual distance is considered to be correct. The trainer may use any threshold he desires.

Results

The distances covered were based on the minimum- and maximum distances defined in Section 6.1.2, which was $30cm$ and $300cm$. Initially the trainer set $X = 10cm$, but the subject expressed concern when he found out that he would need to memorize $\frac{300-30}{10} = 27$ data points. So the trainer changed it to $X = 30cm$ instead, meaning the subject needed to memorize 9 data points only.

The training exercise went as follows:

1. Roughly fifteen minutes, as the subject needed to concentrate and spent time to memorize the tick frequencies. Especially the ticks at distance limits.
2. Took five minutes. The trainer had readied a set of twenty random distance samples to input for the subject to guess beforehand. The data can be seen in Table 6.2.3. For the threshold value the trainer decided that $30cm$ would be an alright choice when calculating the accuracy, meaning the subject should have been able to guess correctly each time if he had memorized the 9 data points, and could spot the difference in frequency. Thus the accuracy was calculated to be 70%, a rather modest result.

Table 6.2.3: The distance results in centimeters after performing step two of the distance training exercise.

Sample	Real [cm]	Subject [cm]	Difference [cm]
1	37	30	7
2	270	200	70
3	130	110	20
4	200	200	0
5	145	150	5
6	300	300	0
7	110	110	0
8	30	55	25
9	47	30	17
10	72	64	8
11	210	170	40
12	290	300	10
13	147	100	47
14	180	177	13
15	47	100	53
16	160	120	40
17	92	80	12
18	39	30	9
19	165	135	30
20	52	90	38

6.2.4 Testing distance recognition

In this test we will get to see how good the subject is at determining distance based on the frequency of the ticks that are being generated by the prototype based on what the ZED Mini stereo camera focuses on. Since we are only interested in the ticks here, the color sound signals are turned off by setting the conditional disable symbol 'no_color_sound' in the software project to 'true'.

The task given to the subject is that he correctly guesses the distance between him and an object while sitting still. The object will be placed at random distances away from the subject, and must reside within the defined minimum- and maximum distance range for the prototype.

Testing preparation

Perform the test in a quiet room that spans atleast the length of the maximum distance the software can handle. Place tables that can cover the same distance in addition to the minimum distance. Then go on to find a light, cube-like object of atleast $1m$ in height that can stand upright on a flat surface without falling over. Mark the vertical center of the highest flat side of the object with black tape.

Lay the tables next to each other so that they form a line that cover a length of the maximum defined distance plus the minimum distance. Place a chair at the end of the tables so that it is perpendicular to the table. From the end of the table where the subject is sitting, fasten the start of a tape measure, then measure up to the maximum distance and more. Fasten the end of the tape measure as well so that it does not go anywhere if touched. Place a table next to the subject's chair. This is where the laptop will reside. Subject sits down on his chair, is instructed by the trainer to stare straight ahead towards the wall, and then mounts the prototype that is already connected to the laptop onto his head.

How to conduct the test

The trainer places the object down one of the marked areas so that the marked flat side is pointed towards the subject. Try to make no sound when placing the object or walking, as this will give hints to the subject on how far away you are. The trainer notes down the actual distance, and the distance that the prototype software estimates. Trainer then asks the subject to guess the distance, and notes down the reply. This is done twenty times. When complete, for all twenty samples, calculate the subject's accuracy against what the software estimated the distance to be. Any guess within a certain threshold of the software distance is considered to be correct. The trainer may use any threshold that he so desires as long as its justified.

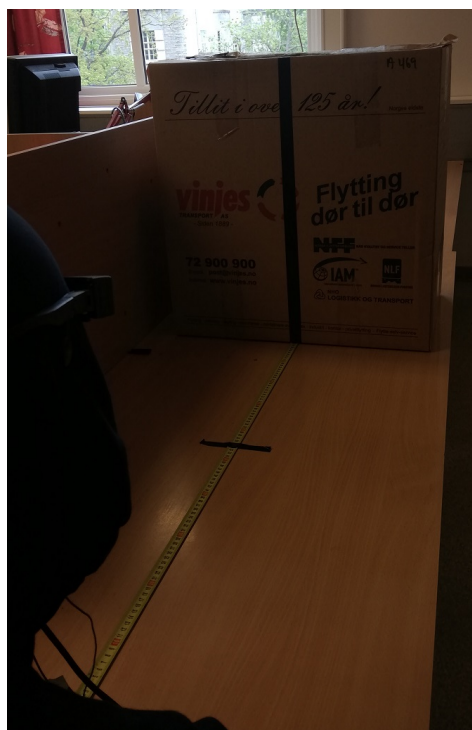


Figure 6.2.3: What our setup looked like for the distance recognition testing.

Results

Our setup can be seen in Figure 6.2.3, where the head of the subject was guided by the trainer so that box was in the center of the prototype vision. The trainer also made sure that the prototype was roughly above the start of the table to make sure the estimated distances were more true. In Figure 6.2.4 you can see what prototype saw when the box was placed at the maximum distance of 300cm. Notice that the software has estimated a distance that is 30cm off the real value.

The results of the tests can be seen in Table 6.2.4. We choose the threshold to be the same as the one used for the training; 30cm. This indicates that the subject correctly guessed the estimate given by the prototype sixteen out of twenty samples, meaning he had an accuracy of 80% with the threshold chosen, which actually shows a slight improvement when compared against the training accuracy of 70% that was calculated in Section 6.2.3. The prototype seemed to reliably estimate distance within 30cm to 300cm, with an accuracy of 65% within a margin of 5cm.

There were potential sources of disturbance however, such as audible sound being emitted whenever the box was being moved, which the subject could have unconsciously picked up on when guessing the distance. This was an unwanted side effect that lead to the test results being inconclusive in determining whether it was the tick frequency alone that had helped aid the subject. This prototype is meant for the visually impaired however, and they will without doubt make good use of all kinds of real sounds to determine how far away something is, so the results were thankfully not entirely without merit. We therefore conclude that the prototype has done well in fulfilling its purpose of giving useful distance estimates to the user, but that there were definitely problems, especially when estimating longer distances. A potential improvement to these faulty estimates are later elaborated on in Section 6.3.3.

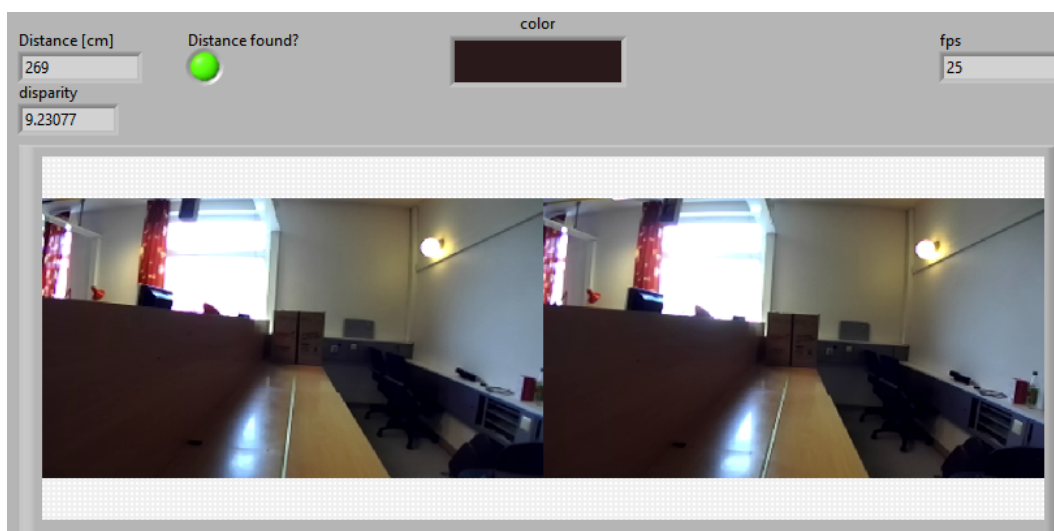


Figure 6.2.4: The point-of-view for the prototype when the box was placed at 300cm.

Table 6.2.4: The distance results in centimeters after having done the distance recognition test.

Sample	Real [cm]	Software [cm]	Diff. Real & Software [cm]	Subject [cm]	Diff. Software & Subject [cm]
1	30	32	2	60	28
2	140	148	8	140	8
3	240	230	10	200	30
4	300	275	25	280	5
5	190	188	2	150	38
6	170	172	2	160	12
7	130	132	2	100	32
8	90	91	1	55	36
9	160	162	2	150	12
10	50	51	1	40	11
11	210	200	10	180	20
12	230	225	5	200	25
13	170	170	0	145	25
14	80	80	0	60	20
15	130	130	0	120	10
16	290	275	15	300	25
17	200	194	6	190	4
18	270	265	5	265	0
19	300	273	27	240	33
20	70	68	2	50	18

6.2.5 Testing both distance and color recognition

For this test we want the subject to be able to utilize the distance and color estimated by the prototype to solve a problem without receiving help from anyone else. It is important that the problem can only be solved if the subject utilizes the prototype, at the assumption that he is blind. He is allowed to use touch to orient himself in this test, but the test is made so that he will still need the prototype to solve it. The point of test is to prove that the prototype can be used to solve a 'impossible' problem for a completely blind person.

The subject is given the task of finding a balloon of a certain color in a room, which can either be white, red, yellow, teal, and blue. There will be five balloons of equal size randomly fastened to a table with tape, and they will all have a unique color that is one of the aforementioned colors. The balloons should be indistinguishable through touch. Before doing the test the subject is given fifteen minutes to memorize the color signals given by each balloon color.

Testing preparation

The test will be done in a room that is somewhat square and quiet, while covering at least an area of $80m^2$ so that there is plenty of space for the subject to move in. The table of balloons is placed in the opposite side of where the room entrance is, and it is assumed that the subject knows about the location of the table before the test.

The element of surprise in this test are the obstacles that will be placed at random positions in the room, which could be stacks of chairs or other tall objects. While these obstacles are placed, the subject must be outside the room so that he can not discern where they have been placed beforehand. The trainer will be determining the obstacle placements.

This test also requires an assistant trainer that will lead the subject into the room after a signal has been given by the trainer that the test is ready.

How to conduct the test

The subject will find himself at the entrance of the room, and will then be told by the trainer what balloon of certain color needs to be identified. The trainer starts a stopwatch that runs until the test is complete. The subject now has to find the table of balloons and make the guess. After the subject has given a guess, the trainer stops the timer, then write the time spent and if the subject guessed correctly.

Results

Since our subject was not blind, his eyes was covered with a dark hood that could not be seen through. The computer connected to the prototype ran the software, and was placed in a backpack for the subject to wear. The prototype rested over the hood at the front of the subject's skull. The assistant trainer was selected to be Jon Petter, a member of this project.

We decided to the the test three times, and for each one the placement of the balloons on the table was randomized. We added progressively more and more obstacles for each test to make it more difficult for the subject to get to the table of balloons.

For the first test the subject was asked to identify the red balloon. It took 2 minutes and 12 seconds, and the subject guessed correctly. A picture of the test in action can be seen in Figure 6.2.5.



Figure 6.2.5: The subject looking for the red balloon in the first test.

In the second test the subject was asked to identify the blue balloon. It took 2 minutes and 47 seconds, and the subject guessed correctly. A picture of the test in action can be seen in Figure 6.2.6.



Figure 6.2.6: The subject looking for the blue balloon in the second test.

In the third and final test the subject was asked to guess the white balloon. This time it took 6 minutes and 6 seconds because there were significantly more obstacles, but the subject still guessed correctly. A picture of the test in action can be seen in Figure 6.2.6.



Figure 6.2.7: The subject looking for the white balloon in the third test.

The subject did very well on the first two tests in terms of time, but then he got disoriented in the third test and ended up getting lost in the room. It ended up with him finding the walls and following them at a slow pace until he found the table of balloons. He did however correctly guess right balloon for each test. In terms of probability, there was a $(\frac{1}{5})^3 = 0.8\%$ chance of this happening if he guessed randomly, and therefore this was rather pleasing results. The subject also pointed out that several times during the tests he could from the tick frequency 'know' that there was something close right in front of him, and that he was therefore able to avoid it by going around it without needing to bump into it or touch it. This behavior was witnessed by both the trainer and the trainer assistant as well. There were however moments when the subject got confused because of the tick spur bug that is elaborated on in Section 5.3.9.

From the combined tests done we can thus conclude that the prototype helped the subject solve the test. The tick sounds also worked in tandem with touch to aid the subject in navigating the room.

6.3 Room for improvement

Although we had a working prototype, there is definitely areas of importance that could be improved. This section elaborates on changes that could potentially lead to a better prototype, be it by making it more accurate, user-friendly, or robust.

6.3.1 Error handling

The prototype software lacks ways of properly handling errors, which in turn can lead to situations where the user will simply get a cryptic error message. Thus the software needs a error handler that will properly take care of errors if they should occur, so that the software does not crash spontaneously. For a visually impaired person, showing a error in a text box is not helpful, so it is wiser to utilize a error handler function that somehow silences the sonification audio, then plays a unique error message sound signal to notify the user what error came about. This would on the flip-side require that the user is trained in recognizing each unique error sound message.

6.3.2 Color accuracy

At the moment the mean color is calculated based on a percentage p of the center of the center pixel row in the left image, which we will refer to as the center row color method. Although this simple method does give the prototype a color value, there are often situations where it will be flat-out wrong when looking at an object that occupies less space than the percentage p . This can be circumvented most of the time by instead calculating the mean color within a defined area of the left image, which we will call the area color method. This will include the color height area of the object. This will in turn cause the color value to

be more accurate if the object plays into the strength. If the defined area is too large, and the object has for example some green and some red, then the color will be calculated to be yellow, which is incorrect. On the other hand if the defined area is too small, then the color value will be prone to noise in the image, thus it will radically jump between different colors. Thus if this method is utilized, the defined area must be properly optimized so that it is large enough to avoid spurious changes in color values, while also being small enough so that it does not produce wrong color combinations. If the optimization is unsuccessful, then the area color method could turn out to be more inaccurate than the center row color method.

The fact that the prototype software only utilizes the left image at the moment comes with its fair share of problems. First of all it is not intuitive for the user to need to look at an object with the left side of the face, and not the center. The proposed solution is therefore to calculate the color based on both the left- and the right image. The pros of this is that the user can look directly at the object when examining the color. On the other hand, this will crash with how the distance is estimated, since the distance calculation method uses the left image as the reference, meaning if the user looks directly towards an object, the distance for whatever is in the center of the left image is calculated. The distance calculation method was elaborated on in Section 5.1.

Another, but more obvious solution for increasing the accuracy of the color extraction is by utilizing a stereo camera that reproduces colors better. Although this would require added support in the prototype software for the new stereo camera, not to mention a new 3D design for the housing.

6.3.3 Distance range, stability and accuracy

As demonstrated in Section 5.2.2, the distance accuracy on longer ranges can be improved by capturing stereo camera frames with a higher resolution. However, accuracy alone is not the most inherent room for improvement concerning the distance function. The stability, and the systems interaction with the user is just as important. Better stability could possibly be achieved by developing more complex methods for image data analyzing.

6.4 3D- Design and prototype

During our designing process, we hit three different obstacles. The first obstacle was choosing a reference human head, we simply measured our heads and our noses to get a design that would fit us. We ended with 3 different designs for the nose pads, none of them had a universal comfortable fit. To elevate the design, one needs to 3D scan a large selection of people, with different head sizes and shapes of course. The 3D scanned models would aid in creating more organic and comfortable shapes for the nose pads and the temples. The second obstacle was the small selection of available materials, 3D printed PLA and PETG are simply not strong enough to create a minimalist and elegant design.

We recommend that the next iteration should be produced with a nylon-based material. Unfortunately, nylon require a substantial volume of heat to reach it's melting point, most traditional 3D printers are not capable of utilizing this material. Other production methods such as Stereolithographic 3D printers or injection molding may be required. During the research phase we searched vigorously for a compact USB type-c cable that has the right orientation and true USB 3.0 speeds, unfortunately we never found a proper solution. We believe that a custom cable specifically made for this prototype would tremendously help in achieving more comfortable experience.

Chapter 7: Conclusion

7.1 Were the requirements satisfied?

For the first requirement specified, we were asked to compare different technologies used in stereo camera systems. During the research phase we looked into two technologies commonly used for consumer stereo cameras, more specifically IR stereo triangulation and passive triangulation.

When selecting a stereo camera that would fit the requirements made by the customer, we had to make sure that it could be integrated with LabVIEW. This went rather smoothly, since capturing stereo camera frames from the ZED Mini is easily achieved through the use of the NI-IMAQdx module for LabVIEW. Another condition given by the customer was that the stereo camera could reproduce colors well. While the ZED Mini has difficulties with reproducing red colors in both dark and well lit environments, it stands its ground quite well in normal lighting conditions at 450lx. The last criteria for the chosen stereo camera was that it needed to be small enough for it to be integrated into what was at the beginning of this project, referred to as the primary Colorophone design. The customer had no issue with us designing our own housing instead of using the old design, and so we consider this criteria to be satisfied.

The third fundamental requirement was that the prototype could get the distance from a defined point in the form of digital information. The implemented distance calculation method is developed from scratch in LabVIEW, and is proven to give relatively good results considering the simplicity of the method. The prototype testing done demonstrated that the prototype is able to calculate the distance based on stereo vision, relatively accurately, up to about three meters. The accuracy is best at lower distances; which is closely related to the camera resolution used for the prototype.

One of the supplemental requirements was that the existing Colorophone color coding method could be integrated into the prototype software. This was not difficult, since it could be realized by just utilizing the existing color coding method on the left image in the stereo camera frames.

It was asked of us to expand Colorophone into a system that could provide the user with visual echolocation. Some of the coding work had already been done for this one, in conjunction with the third fundamental requirement, so it was almost just a copy and paste job into

the software application we developed for the prototype. Distance information was conveyed in the form of tick frequencies. When testing the prototype with a subject that had been trained to recognize the tick sounds, the subject correctly guessed within a margin of 30cm, 70% of the distances. Although a rather large margin, it still worked as a concept, and a potential interesting future project would be to test how much training would be needed for a subject to get a almost perfect accuracy with a small margin of 5cm.

7.2 Are stereo cameras viable for Colorophone?

There is much to be desired when imagining how one could cram a physical scene of vibrant colors and objects into auditory signals, but when thinking in terms of colors and distance alone, our prototype shows that there is definitely usefulness in utilizing stereo cameras to provide distance information. In spite of its shortcomings, there is much room for improvement of the prototype, be it more reliable distance estimation through clever algorithms, or a more robust prototype housing.

7.3 Future projects

Stereo cameras opens up a window to a world of possibilities by taking inspiration from how humans visually perceive light, and we believe it is a step in the right direction for Colorophone. Although stereo cameras as gadgets are only in their developing stage, there is no doubt that they will see more widespread use in the coming future. Not only will we see increased usage of AR devices that blurs the line between the virtual world and reality, but also stereo vision for the smartphones that have conquered the world in just a decade alone. Considering the fact that Colorophone has already developed an application for smartphones that lets users sonify colors; the idea of providing visual echolocation with AR headsets or smartphones through Colorophone might not just be a pipe dream.

Bibliography

- [1] D. Osiński, “A sensory substitution device inspired by the human visual system,” 2018.
- [2] “National Instruments, LabVIEW homepage,” <http://www.ni.com/en-no/shop/labview.html>, Last checked (28.05.19).
- [3] “CLRF Research (Unreleased),” <https://play.google.com/store/apps/details?id=com.colorophone.app.research>, Mobile Application, Google Play, Published May 2017.
- [4] A. T. Myhr and E. Sanden, “NTNU Bachelor-Thesis: Wearable eye-tracking system,” 2017.
- [5] R. Cong and R. Winters, “How Does The Xbox Kinect Work,” <https://www.jameco.com/jameco/workshop/howitworks/xboxkinect.html>, (Last read: 28.05.2019).
- [6] “Tesla Inc: Tesla Autopilot,” <https://www.tesla.com/autopilot>, (Last read: 28.05.2019).
- [7] “Intel Inc: Intel® RealSense™ Camera R200 specifications,” <https://ark.intel.com/products/92256/Intel-RealSense-Camera-R200>, (Last read: 28.05.2019).
- [8] Tenney and Matthew, “Microsoft Kinect - Hardware,” <http://gmvcast.uark.edu/scanning/hardware/microsoft-kinect-resourceshardware/>, 2012, (Last read: 28.05.2019).
- [9] “Orbbec 3D: Orbbec Astra Mini and Astra Mini S,” <https://orbbec3d.com/astra-mini-series/>, (Last read: 28.05.2019).
- [10] “e-con Systems: Tara - USB 3.0 Stereo Vision Camera,” <https://www.e-consystems.com/3D-USB-stereo-camera.asp>, (Last read: 28.05.2019).
- [11] “Ensenso Inc: N30 and N35 Stereo 3D Cameras,” <https://www.ensenso.com/portfolio-item/n3x/>, (Last read: 28.05.2019).
- [12] “ASUSTeK Computer Inc: Xtion PRO LIVE,” <https://www.asus.com/us/3D-Sensor/Xtion.PRO.LIVE/>, (Last read: 28.05.2019).
- [13] “Intel Inc: Intel® RealSense™ Camera D415 specifications,” <https://ark.intel.com/products/128256/Intel-RealSense-Depth-Camera-D415>, (Last read: 28.05.2019).
- [14] “Intel Inc: Intel® RealSense™ Camera D435 specifications,” <https://ark.intel.com/products/128255/Intel-RealSense-Depth-Camera-D435>, (Last read: 28.05.2019).
- [15] “Stereolabs: Meet ZED,” <https://www.stereolabs.com/zed/>, (Last read: 28.05.2019).

- [16] “Stereolabs: Meet ZED Mini, the world’s first camera for mixed-reality,” <https://www.stereolabs.com/zed-mini/>, (Last read: 28.05.2019).
- [17] “Wikipedia on Binocular Vision,” https://en.wikipedia.org/wiki/Binocular_vision, Last checked (28.05.19).
- [18] Brian A. Wandell, “Foundations of Vision, Chapter 10: Motion and Depth,” https://foundationsofvision.stanford.edu/chapter-10-motion-and-depth/#Binocular_Depth, Last checked (28.05.19).
- [19] “Wikipedia: Article about binocular disparity,” (Last read: 28.05.2019). [Online]. Available: https://en.wikipedia.org/wiki/Binocular_disparity
- [20] “Wikipedia: Article about the correspondence problem,” (Last read: 28.05.2019). [Online]. Available: <http://en.wikipedia.org/wiki/Correspondenceproblem>
- [21] “Wikipedia: Article about Cross-correlation,” <https://en.wikipedia.org/wiki/Cross-correlation>, (Last read: 28.05.2019).
- [22] S. Dorodnicov, “The basics of stereo depth vision,” <https://realsense.intel.com/stereo-depth-vision-basics/>, (Last read: 28.05.2019).
- [23] “Stereolabs Inc: Depth from Stereo,” <https://www.stereolabs.com/docs/depth-sensing/>, (Last read: 28.05.2019).
- [24] “How does part orientation affect a 3D Print?” <https://www.3dhubs.com/knowledge-base/how-does-part-orientation-affect-3d-print>, (Last read: 28.05.2019).
- [25] “Best 3D Printer Filament Types Guide of 2019,” <https://www.allthat3d.com/3d-printer-filament/>, (Last read: 28.05.2019).
- [26] “Wikipedia: Article about the focal length,” (Last read: 28.05.2019). [Online]. Available: https://en.wikipedia.org/wiki/Focal_length

Definitions

ABS Acrylonitrile butadiene styrene. A common thermoplastic polymer, widely used in the 3D printing community for durable parts.

AR Augmented reality. Adds the possibility to combine information from the physical world with virtual data, a good example is the use of heads-up displays in cars that projects vital information such as speed on to the windshield of the car.

Baseline The fixed distance between the two lenses in a stereo camera module.

Binocular disparity The difference in location of objects within the left and right image in a stereo vision system [19]. It can also be referred to as disparity in this thesis.

CAD Computer-Aided Design. The application of computers to create and analyze a design in a digital space.

dB Decibel. A measurement unit that expresses the ratio of a quantity on a logarithmic scale.

DBL Double data type. A double-precision floating-point format that occupies 64 bits in memory, and represents a fractional number.

eMMC Embedded MMC or embedded MultiMediaCard. A type of storage unit that is solid, i.e no moving parts, has decent speeds and is used in smartphones, other consumer electronics, and some budget friendly laptops such as ultraportable laptops.

FFF Fused Filament Fabrication. A 3D printing process that continuously feeds filament through a hot end and deposits, and builds the 3D model layer by layer.

FIFO First In First Out. Normally used when describing how a queue handles elements. The first enqueued element is the first one that is dequeued.

Filament Thermoplastic feedstock for FFF 3D printers. A long piece of plastic string on a spool that is fed into the printer. There are many different types of plastics used with different properties and sizes, the two standard sizes are 1.75mm and 2.85mm in diameter.

Firmware Software installed for hardware, and performs low-level control of the device.

Focal length The distance between the lens and the focal point in an optical system. In a camera, the light sensor is located in the focal point. The relation between a camera's

focal length and it's field of view is inversely proportional [26].

FPGA Field-programmable gate array. A integrated circuit (IC) that give developers the ability of programming low-level hardware.

GUI Graphical User Interface. Is used to describe the interface that users interact with in software.

IR Infrared Radiation. Electromagnetic radiation invisible to the human eye, with longer wavelength than visible light.

LPF Low-pass filter. A filter that only passes frequencies that are lower than a defined cutoff frequency.

lux abbr. lx. A unit that measures the luminous flux per unit area.

NI-IMAQdx A driver package that is included with the NI Vision Acquisition Software module, that aid in setting up and utilizing cameras in LabVIEW.

OTG On The Go. A specification that allows a USB device such as smartphones and tablets to act as a host, allowing other USB devices to be connected to them, such as storage drives, keyboard, and mice.

PET Polyethylene Terephthalate. A plastic commonly used in bottles, filament for 3D printing, and textile products, such as fleece.

PETG Polyethylene Terephthalate Glycol-modified. A glycol modified version of PET, often used as filament for 3D printing.

Pixel abbrv. px. A physical point in a digital image, and contains color information for red, green, and blue.

PLA Polylactic Acid. A biodegradeable plastic made by renewable resources such as corn starch, widely used for 3D printing.

PNG Portable Network Graphics. A lossless file format commonly used to represent images utilizing transparency.

sRGB standard Red Green Blue. A default color space commonly used in monitors and printers.

Stringing A unwanted side effect that can occur when 3D printing, where a string of residue is left between gaps in a 3D print.

ToF Time of Flight. A technique used to estimate distance based on stereo camera frames.

TPU Thermoplastic Polyurethane. A common form of elastomer with a very high flexibility and durability, widely used for 3D printing when the print needs to be flexible.

RGB an additive color model that incorporates red, green, and blue as elementary colors to represent more complex colors.

RGBW abbrv. for Red, Green, Blue and White.

SDK Software Development Kit. A set of tools specifically designed for use in developing software for hardware- or software products.

VR Virtual Reality. Allows the user to interact in a computer created reality with the use of stereoscopic head-mounted display.

Stereo vision A vision system that includes two sets of lenses or eyes. Has the ability to provide depth perception in addition to other visual perceptions.

Stereo camera A camera that consist of two image sensors, and stitches the two images into one image frame.

Appendices

A Hardware

A.1 MT-906 Light Meter

Since light has a big influence on how cameras operate, it was necessary for us to get our hands on a light meter so we could measure how much light was present when comparing the ZED Mini and the Intel RealSense D415 against each other. We therefore acquired a MT-906 Light Meter (see Figure A.1) from a local *Clas Ohlson* store, and was used for measuring the light level when comparing Stereolabs ZED Mini and the Intel RealSense D415. It has a range of $0lx$ to $100000lx$, and a accuracy of $\pm 3\% \text{ rdg} \pm 0.5\% \text{ f.s}$ for light less than $10000lx$, and $\pm 4\% \text{ rdg} \pm 10\% \text{ dgts.}$ for more than $10000lx$. The light meter in question is rather crude, and therefore not the most accurate measuring device, but for our purposes it was good enough to give us an estimated light level.



Figure A.1: A picture of the MT-906 light meter¹.

A.2 Calibrated monitor, EIZO ColorEdge G2730

In order to test how well the stereo cameras read colors, we utilized a specialized monitor, a *EIZO ColorEdge CG2730* (see Figure A.2) that we borrowed from our customer. It is a 27 inch monitor with a 2560x1440 screen resolution that can display a wide range of colors. A typical monitor uses 8-bits to represent colors with values from 0 to 255, but this monitor

¹Picture taken from:
https://www.clasohlson.com/medias/sys_master/9580195872798.jpg

uses 10-bits with a 16-bit look-up table, meaning that the monitor has the capability to display over one billion (1024^3) unique colors, compared to a normal monitor that only has 16 million (256^3). The monitor also comes with a built-in calibration sensor for calibrating the colors².



Figure A.2: A picture of EIZO ColorEdge CG2730³.

²See <https://www.eizoglobal.com/products/coloredge/cg2730>.

³Picture taken from:
https://www.eizoglobal.com/products/coloredge/cg2730/product_photo_01.png

A.3 AAEON UP Board

During the research phase, the *AAEON UP Board* (see Figure A.3) was being investigated as a probable mobile computer that could partner up with the Colorophone glasses. It carried decent specifications for its size, being no larger than a credit card. It came bundled together with the *Intel RealSense R200* stereo camera that the team received from the customer at the start of the project.

The UP Board came with a quad core Intel® Atom™ x5-z8350, 4 GB RAM, and 32 GB of eMMC storage⁴. In addition, 4x USB 2.0, Ethernet, and a USB 3.0 OTG connection was included. The board supported several Linux operating systems, including Windows 10. It was capable of using LabVIEW, but was in the end ruled out due to the lack of peripherals like a monitor, or a keyboard, which would have made it impractical to use for field.



Figure A.3: The AAEON UP Board without the aluminium cooling block and fan attached.⁵

⁴See <https://www.aaeon.com/en/p/up-board-computer-board-for-professional-makers>.

⁵Picture taken from:
<https://www.aaeon.com/en/p/up-board-computer-board-for-professional-makers>

A.4 myRIO

The myRIO (see Figure A.4) is a hardware device developed by National Instruments, and is powered by LabVIEW. RIO stands for Reconfigurable I/O. The myRIO has an onboard FPGA in addition to the processor. The FPGA can be programmed to run specialized tasks very fast, and therefore it is perfect for use in real-time-systems. To use the myRIO as the processing unit in the prototype, the colorophone software architecture must be specially designed for the myRIO. The project was from the start based on using a windows computer as processing unit. The myRIO was not used for the prototype because the amount of time needed to adapt the software with the myRIO was too much.



Figure A.4: The myRIO device by National Instruments.⁶

⁶Picture taken from:
<https://www.digitec.ch/en/s1/product/national-instruments-782693-01-hardware-ni-myrio-19-development-boards-kits-850369>

A.5 HP Pro Tablet 608 G1

It was of interest that we had utilized a lightweight computer capable of running LabVIEW. After discussing this with the customer, the *HP Pro Tablet 608 G1* (see Figure A.5) we thought might be a good option as it was already confirmed by the customer that it had the ability to run LabVIEW⁸. The tablet comes with a quad core Intel Atom® x5-Z8500 processor, 4 GB of RAM, 64 GB of storage, and a 21 Wh battery. In addition, it comes with a single USB-C connection, which is convenient as the *ZED Mini* also has a USB-C connector. It was however discovered, during testing of the prototype, that the tablet performed subpar to what we had hoped for. There was also issues with the audio when running the Colorophone software. A laptop was therefore used, instead of the tablet, to run the software during testing.



Figure A.5: HP Pro Tablet 608 that were used during testing⁷.

⁷Picture taken from:
<https://dustinweb.azureedge.net/image/149256/400/320/hp-pro-tablet-608-g1.jpg>

⁸See <https://support.hp.com/lv-en/document/c04718256>.

A.6 Tripod

When testing and calibrating the distance calculation method with the ZED Mini, we needed a tripod that could hold the stereo camera, be adjustable in height, and most importantly, be stable. The tripod was supplied by our customer, and is a *Velbon DF 60* (see Figure A.6), with movable joints for yaw, pitch, and roll, while also having an adjustable height. The tripod did not have a lot that would let us properly connect the ZED Mini onto it, so a makeshift holder was constructed through 3D printed parts. The custom housing could also hold a measuring tape, which made it possible to measure the distances more easily.



Figure A.6: Tripod with the custom 3D printed holder, and the ZED Mini camera attached.

B LabVIEW code

B.1 main.vi

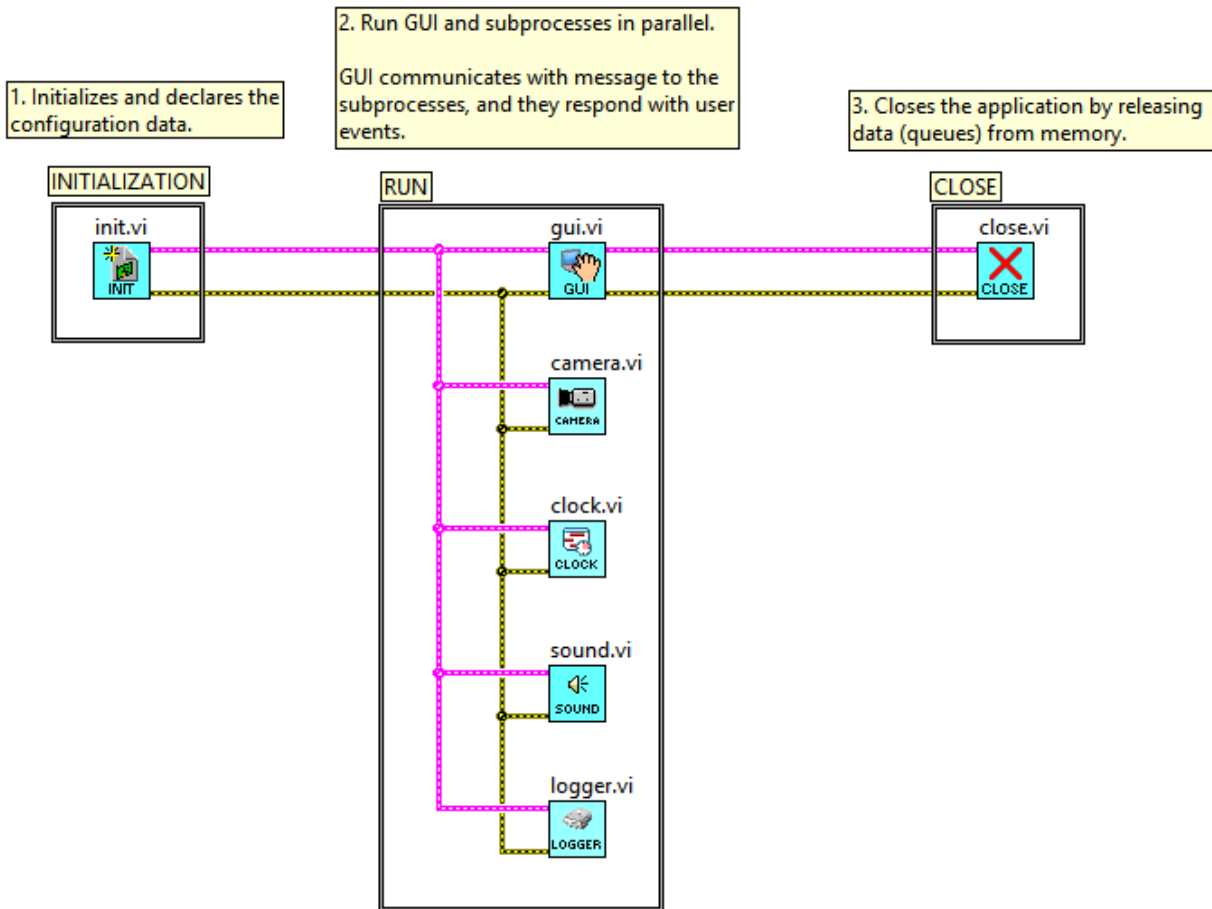


Figure B.1: *This is the .vi that must be run to launch the application.*

B.2 init.vi

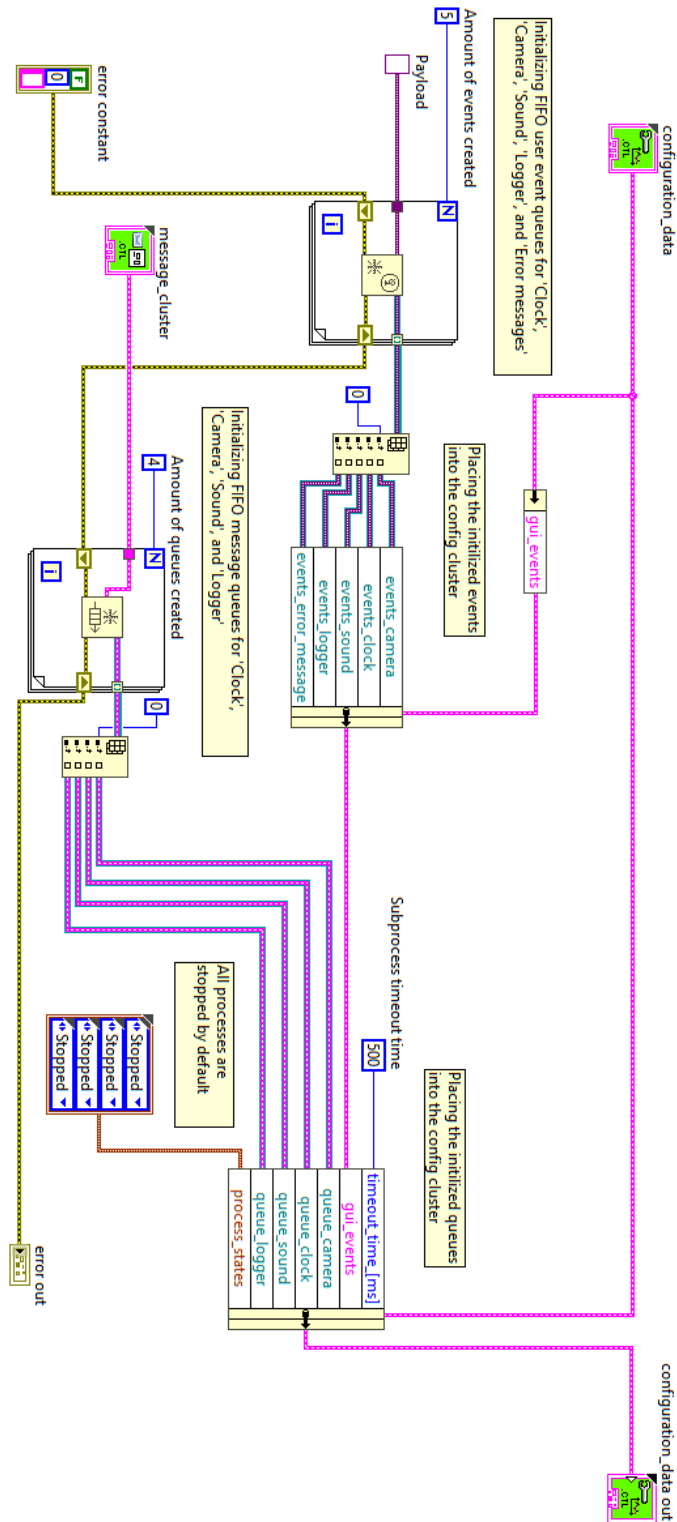


Figure B.2: *Initialization of the application.*

B.3 close.vi

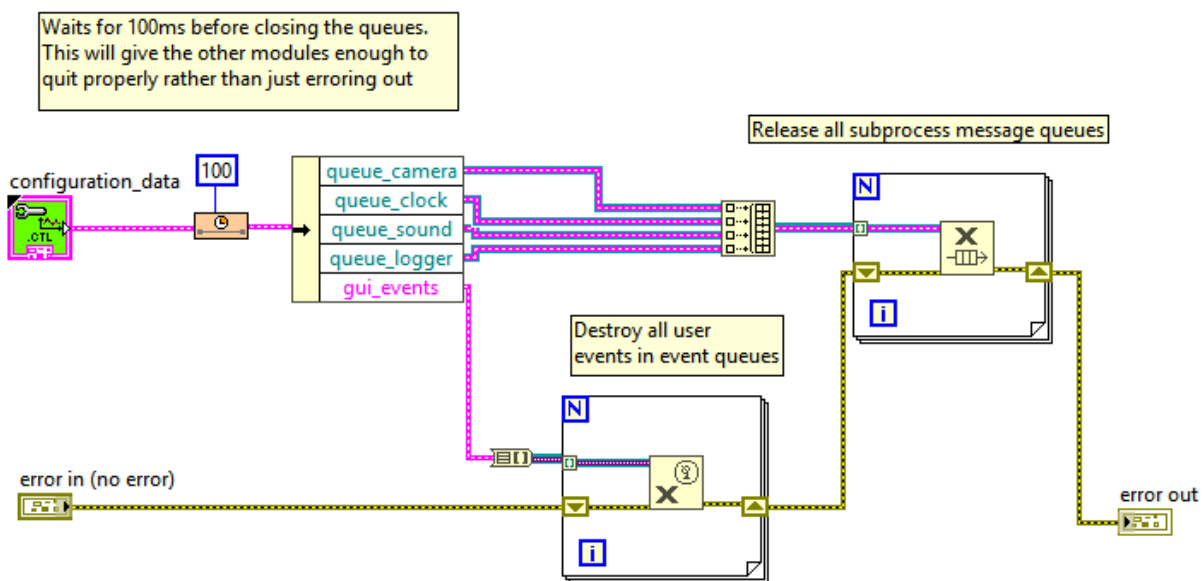


Figure B.3: *Close of the application.*

B.4 color_recognition.vi

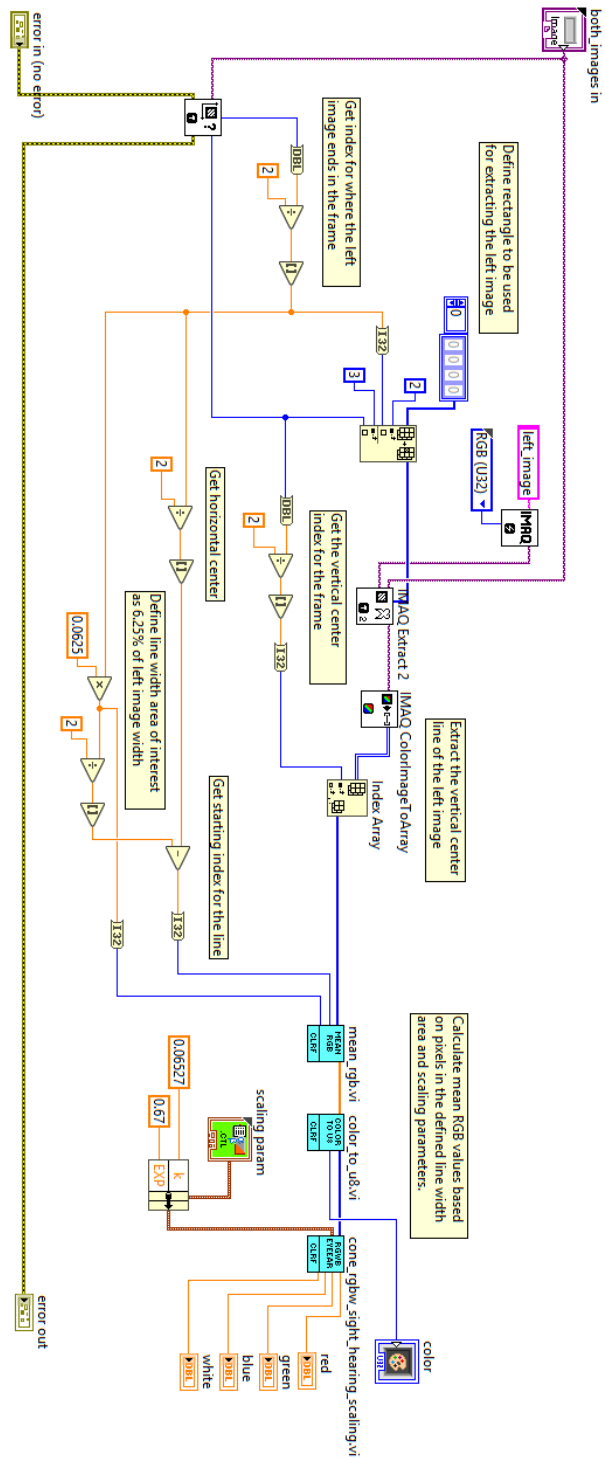


Figure B.4: *Color recognition code.*

B.5 sound.vi

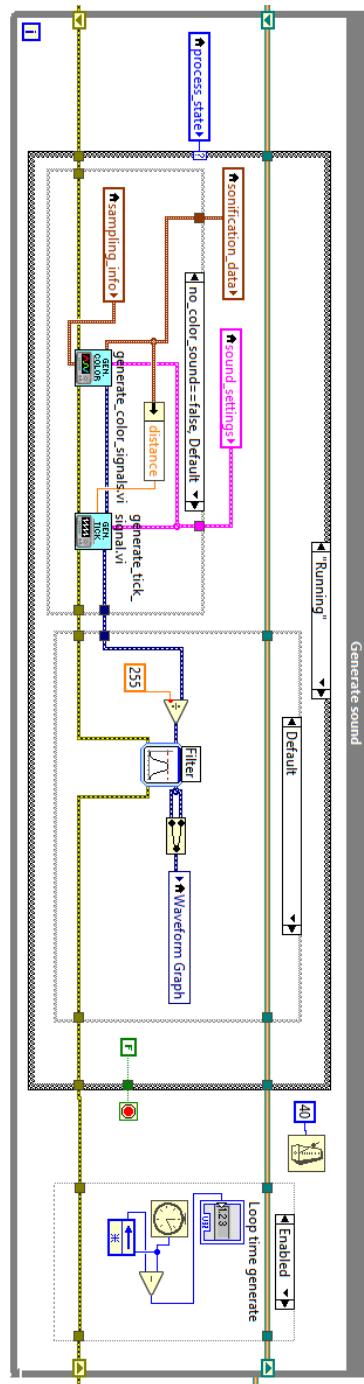


Figure B.5: Loop that is responsible for generating sound signals.



B.6 generate_color_signals.vi

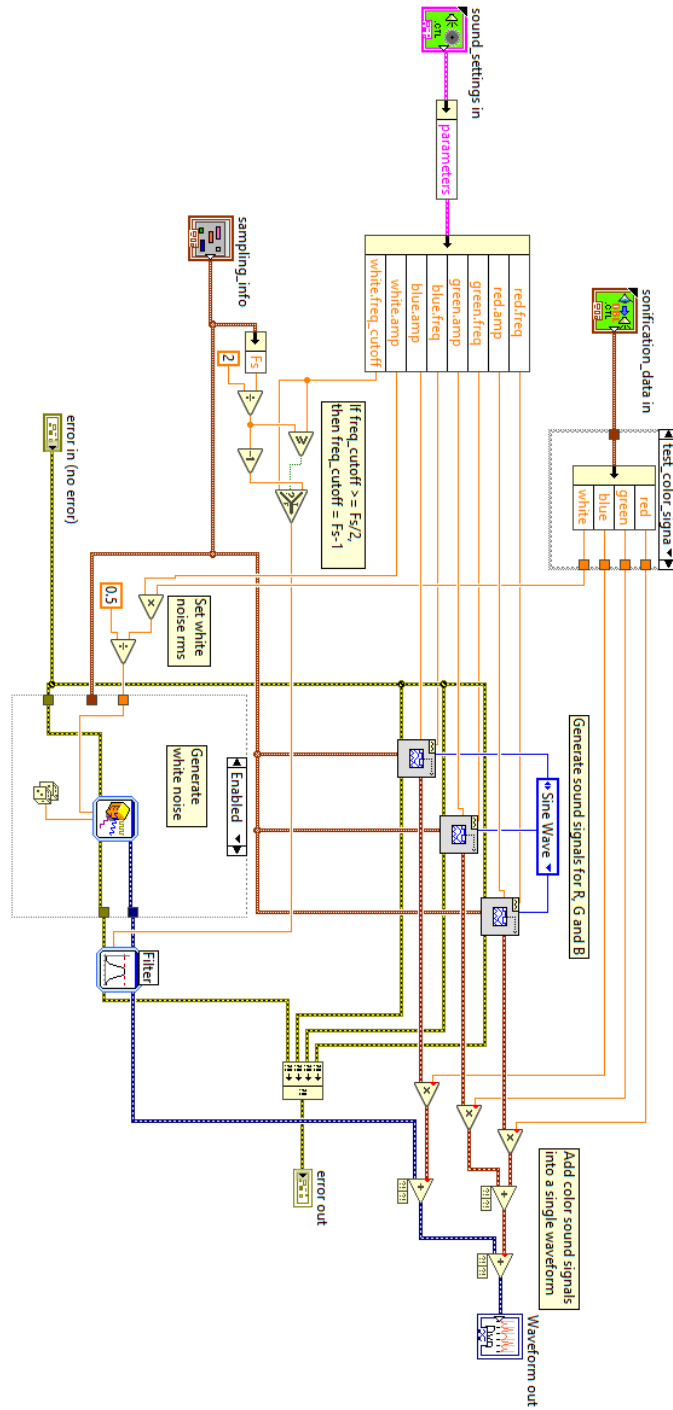


Figure B.7: Code that generates color signals.

B.7 generate_tick_signal.vi

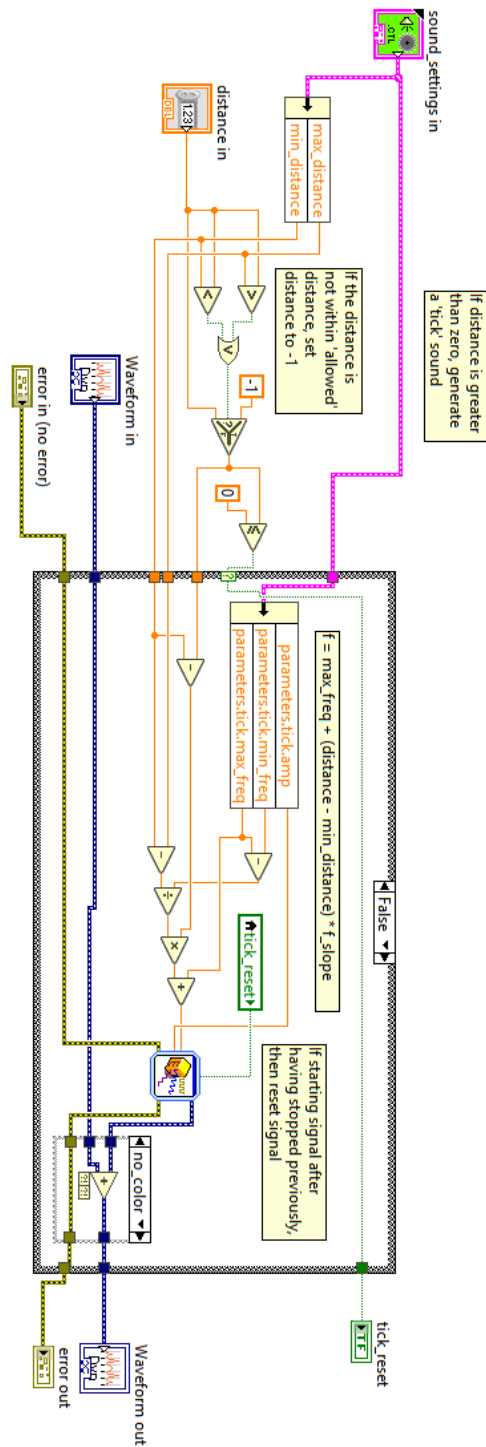


Figure B.8: Code that generates the tick signal.

B.8 distance_ssd.vi

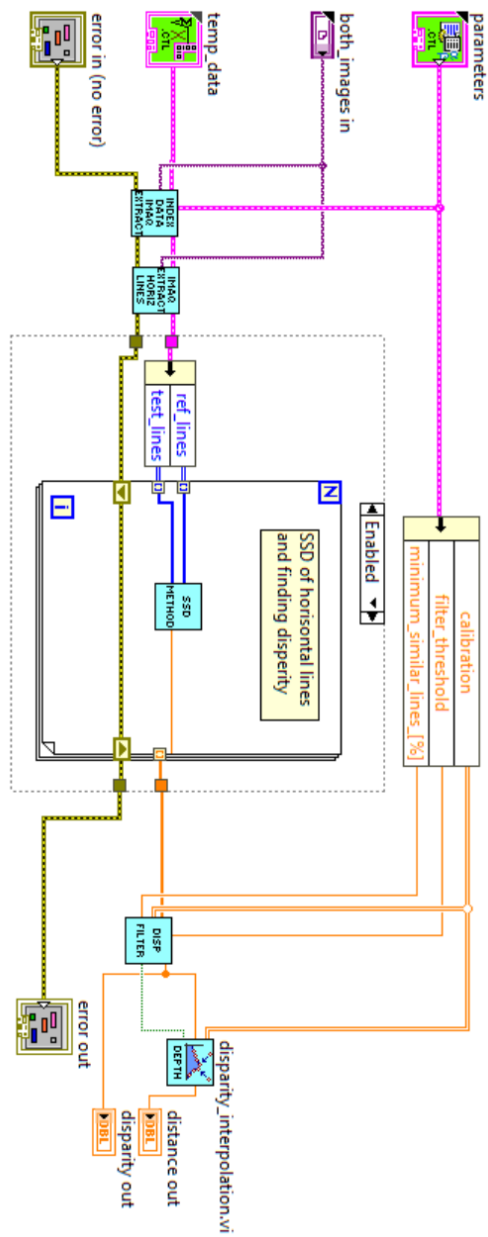


Figure B.9: *Code that performs the distance calculation.*

B.9 IMAQ_extracting_horizontal_lines.vi

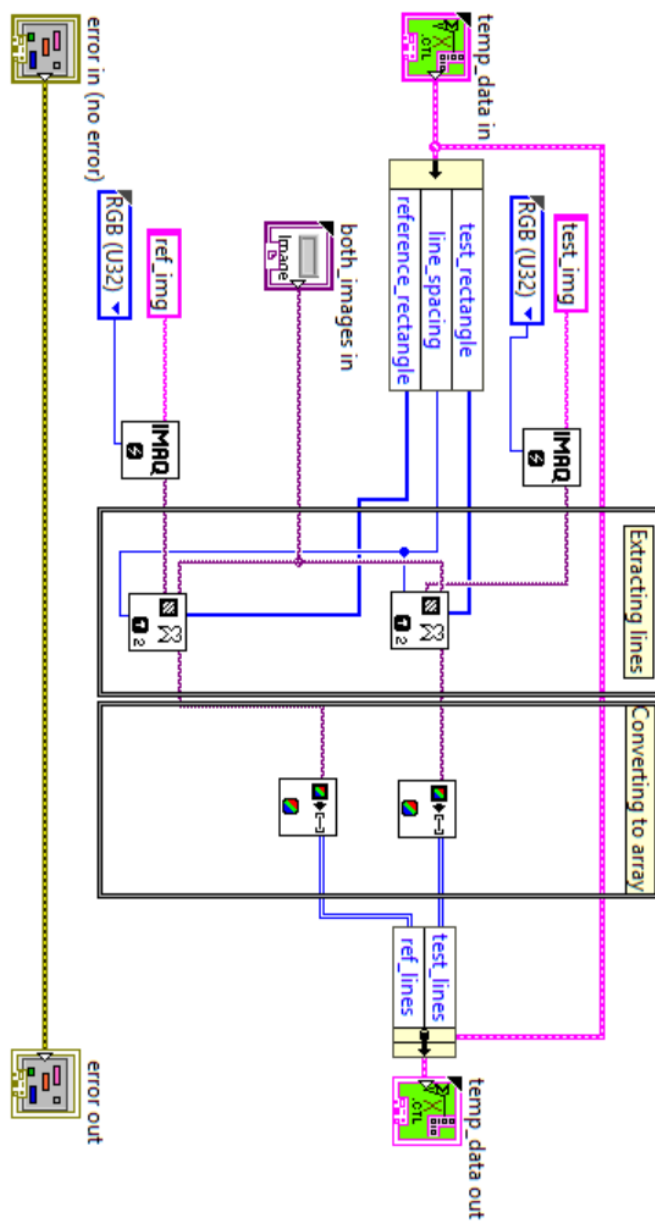


Figure B.10: Code that extracts horizontal lines form camera frame and converts the pixels to array.

B.10 ssd_method.vi

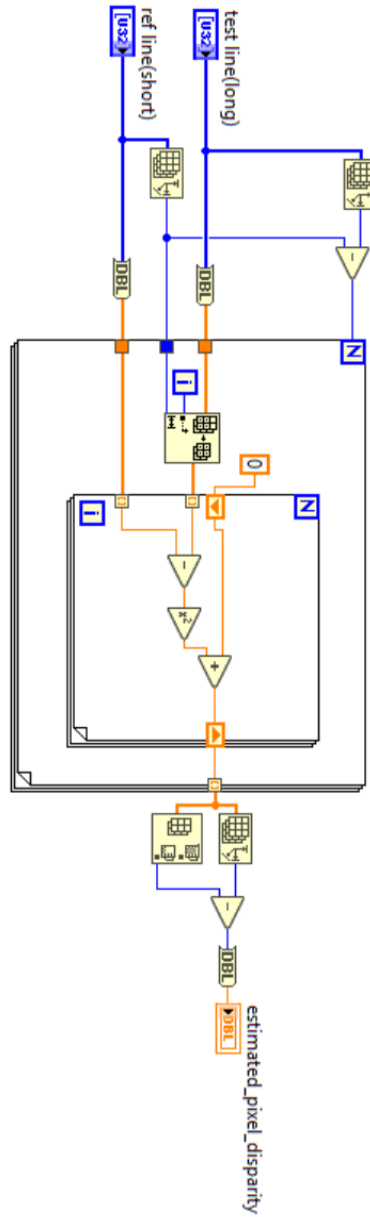


Figure B.11: Code that performs the *sum.sq.diff* method and calculates the disparity corresponding to each set of lines.

B.11 disparity_filtration.vi

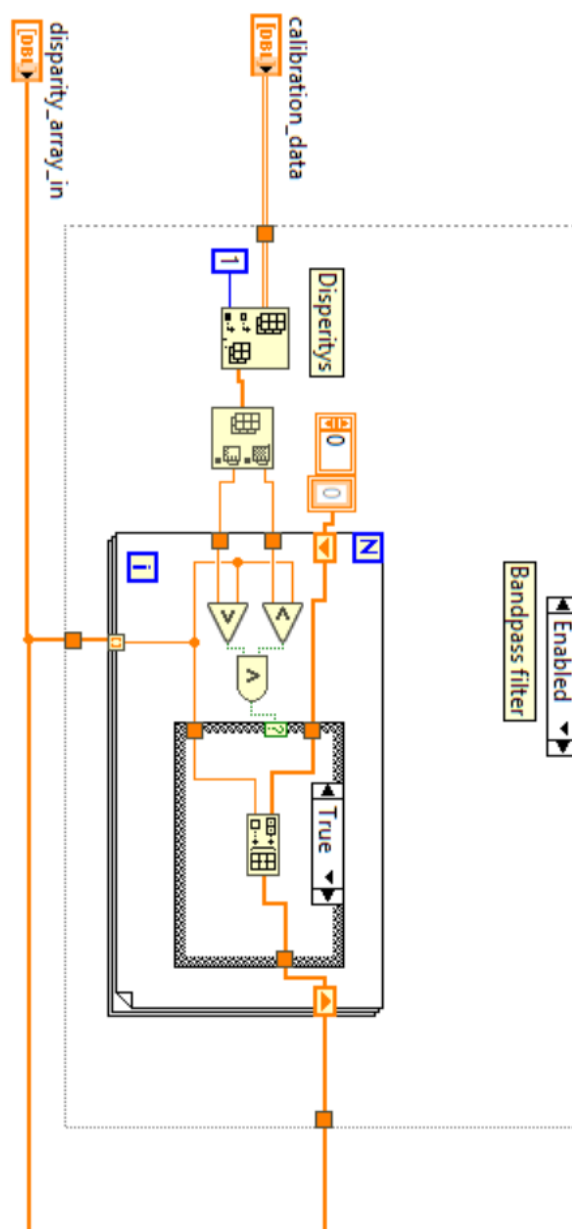


Figure B.12: *Code that filters the array of disparities with a band pass filter to remove disparities outside of range.*

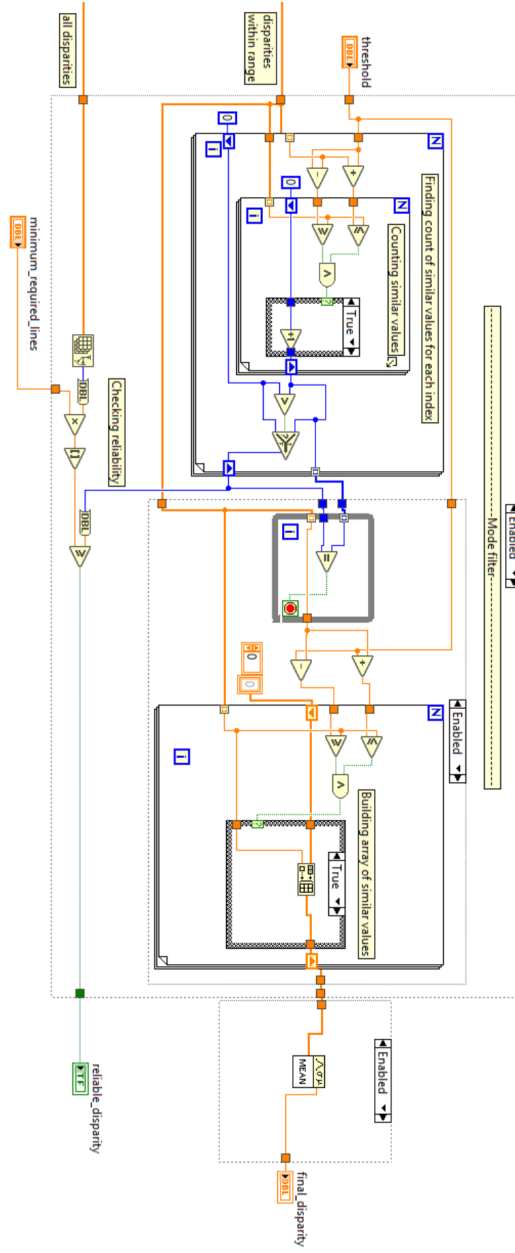


Figure B.13: Code that further filters the array of disparities and passes through the most frequently occurring disparities.

B.12 disparity_interpolation.vi

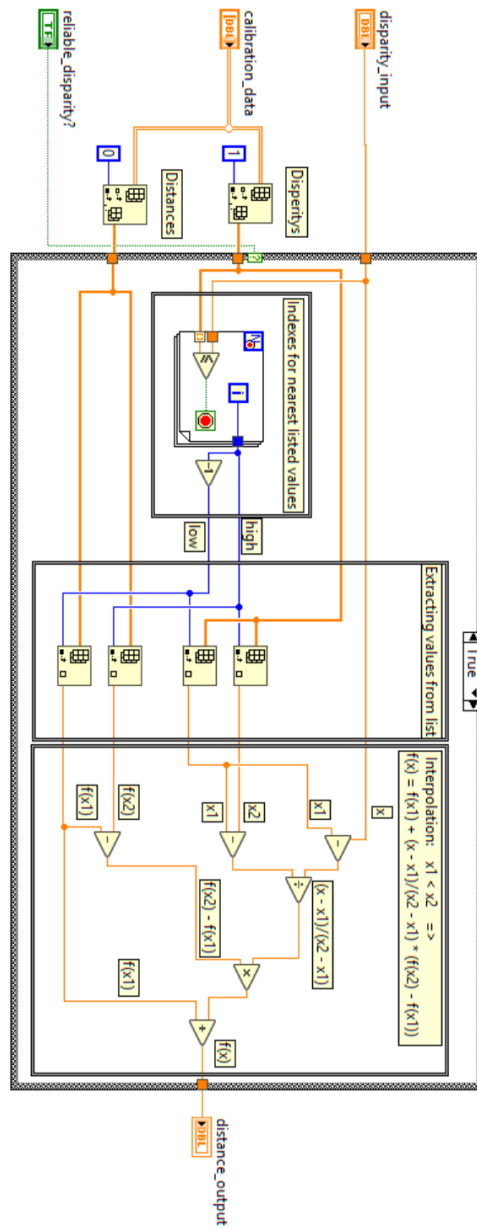


Figure B.14: Code that finds the distance with the use of a look-up table and interpolation.

C 3D-design prototypes

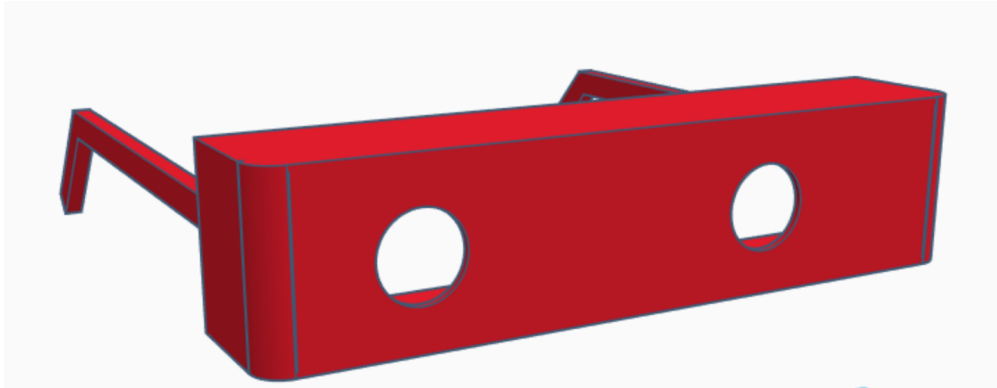


Figure C.1: *What we learned: Lenses must be centered*

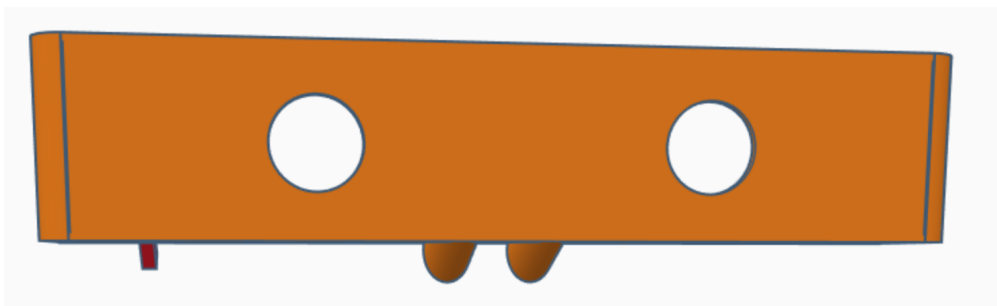


Figure C.2: *What we learned: 1. Less sharp edges. 2. Nose pad must be interchangeable. 3. Temples to short. 4. Temples must be modular*

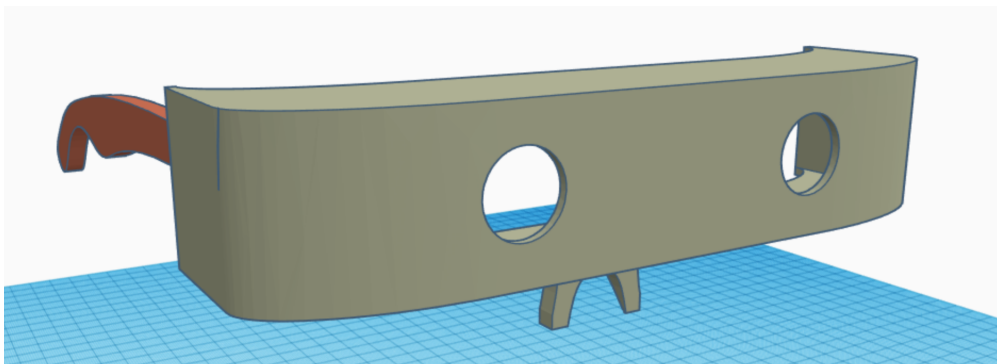


Figure C.3: *What we learned: Temples should be mounted on a collapsible mechanism.*

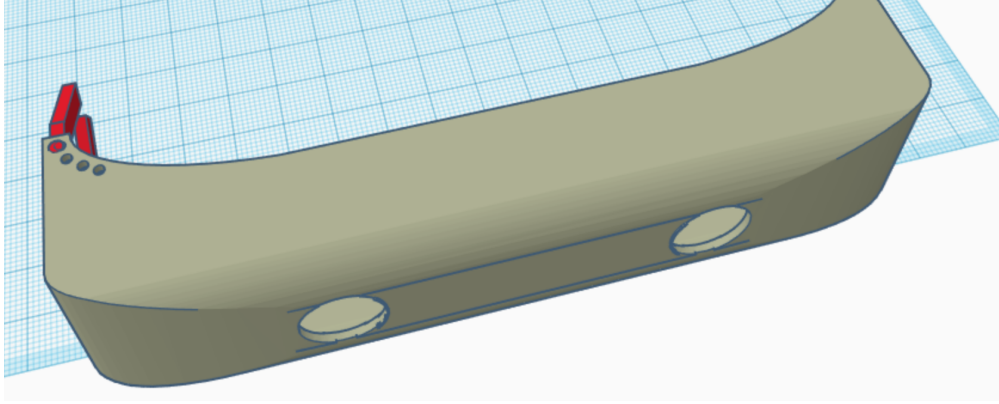


Figure C.4: *What we learned: 1. Upper and lower edges should not be smoothed. 2. Temples mounting mechanism should have two positions, free and fixed. Once the temples are deployed, they should be in the fixed position. They should stay there unless the user deliberately forces the temples back in to the folded “free” position.*

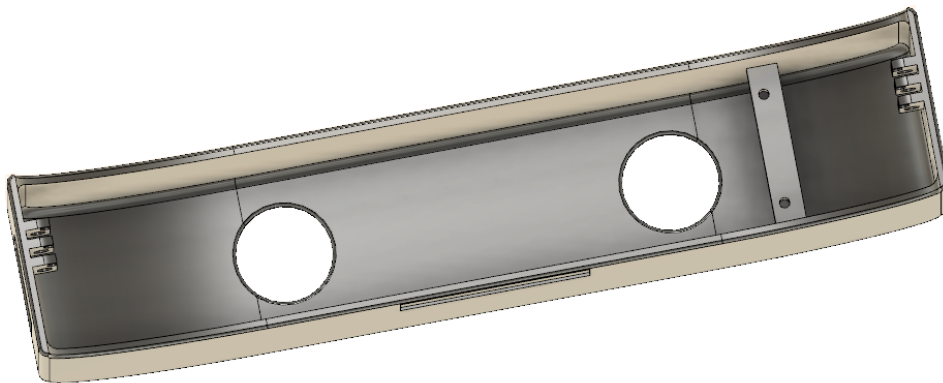


Figure C.5: *What we learned: 1. ZED Mini mounting points should be stronger. 2. Temples collapsing mechanism should be stronger. 3. Setscrews need an ingress for easier entry.*

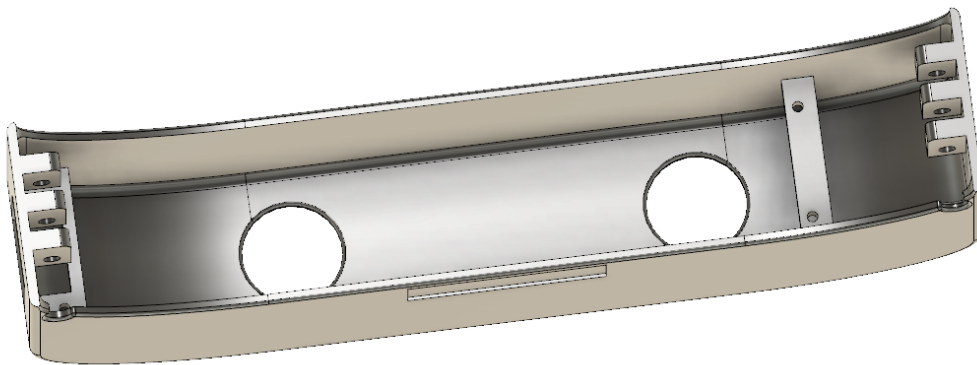


Figure C.6: *What we learned: 1. ZED Mini mounting points should be even stronger. 2. Cut out section for lenses should be smaller.*

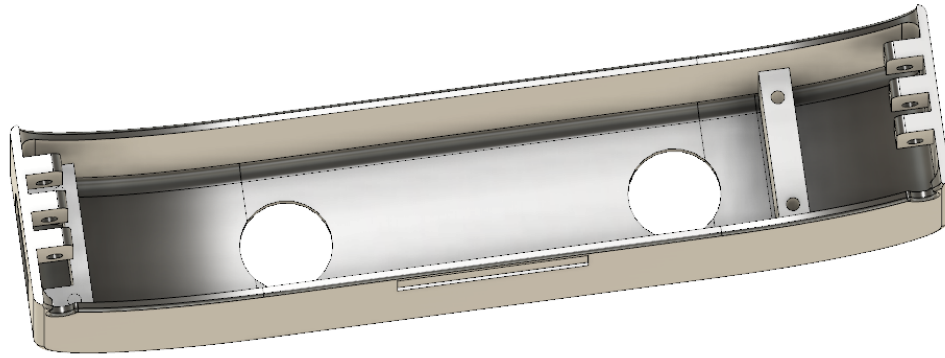


Figure C.7: *What we learned: Right inner side of the housing needs additional strengthening collapsible mechanism.*

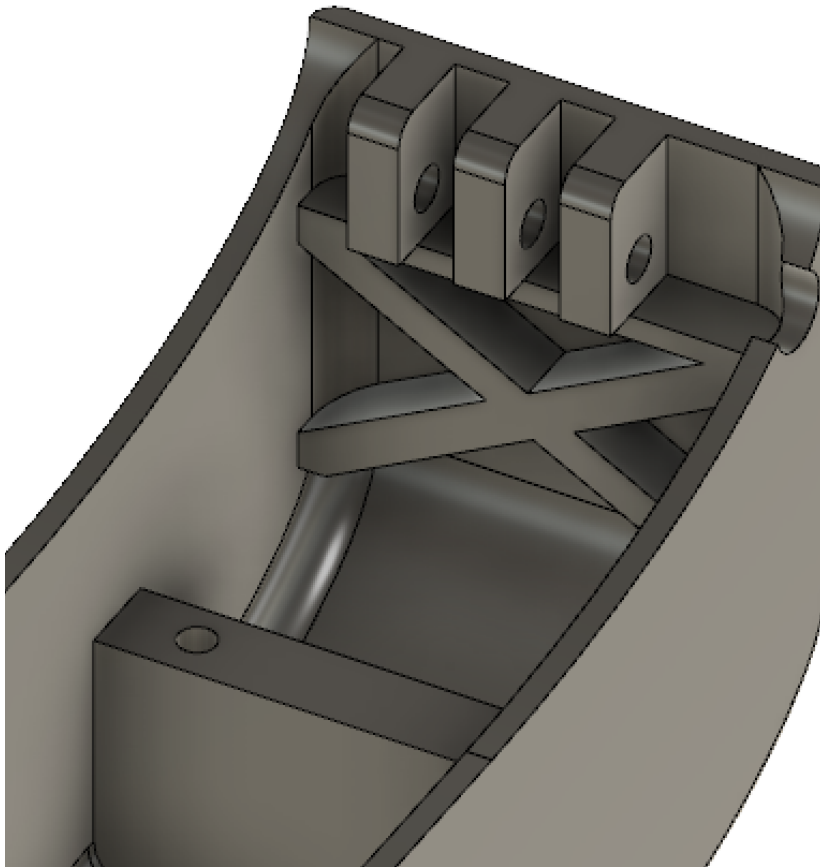


Figure C.8: *Final design*



Figure C.9: *Temples V1*

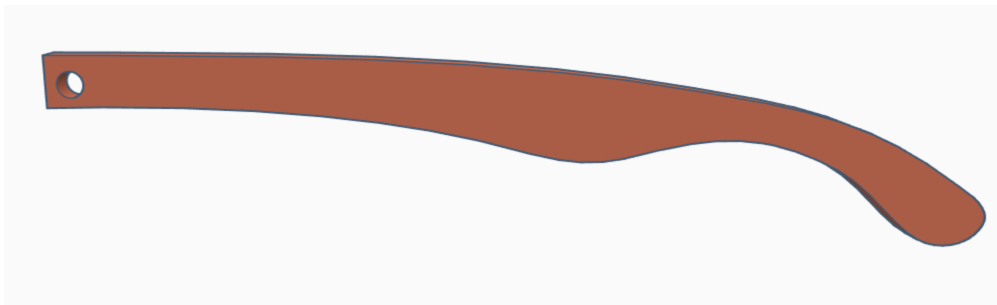


Figure C.10: *Temples V2*

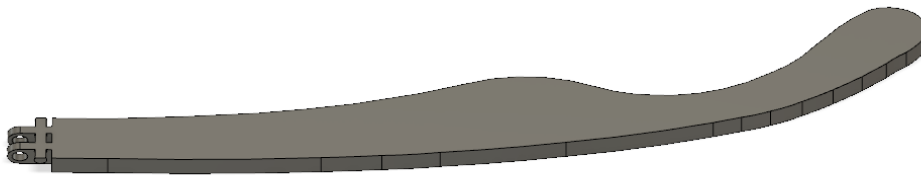


Figure C.11: *Temples V3*

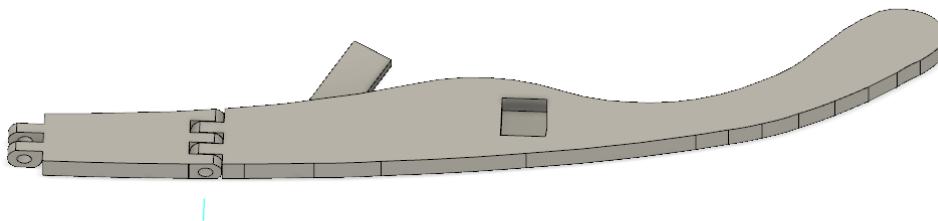


Figure C.12: *Temples V4*

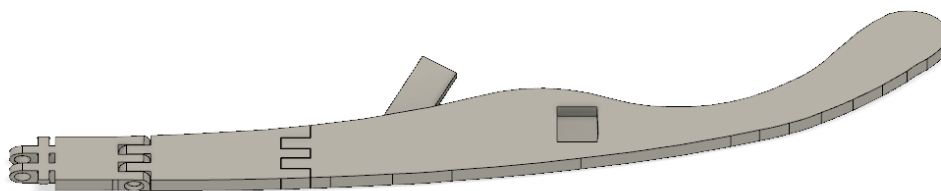


Figure C.13: *Temples V5*

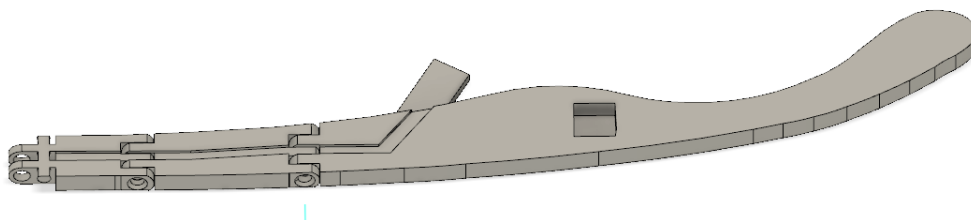


Figure C.14: *Temples V6*

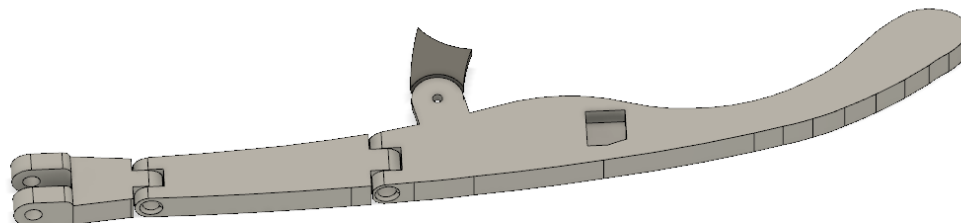


Figure C.15: *Temples V7*

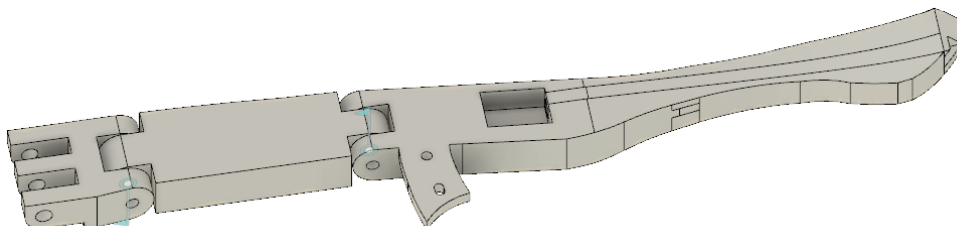


Figure C.16: *Temples V8*

C.1 Computer Free fall simulations

These simulations were done with the Housing injection molded with ABS. The arrow in Figure C.17 and C.18 represents a 5N force that would occur during an impact after a free fall. For the sake of over-engineering we decided to do these simulations as if our prototype weighed 500grams, the actual weight is about a third of that.

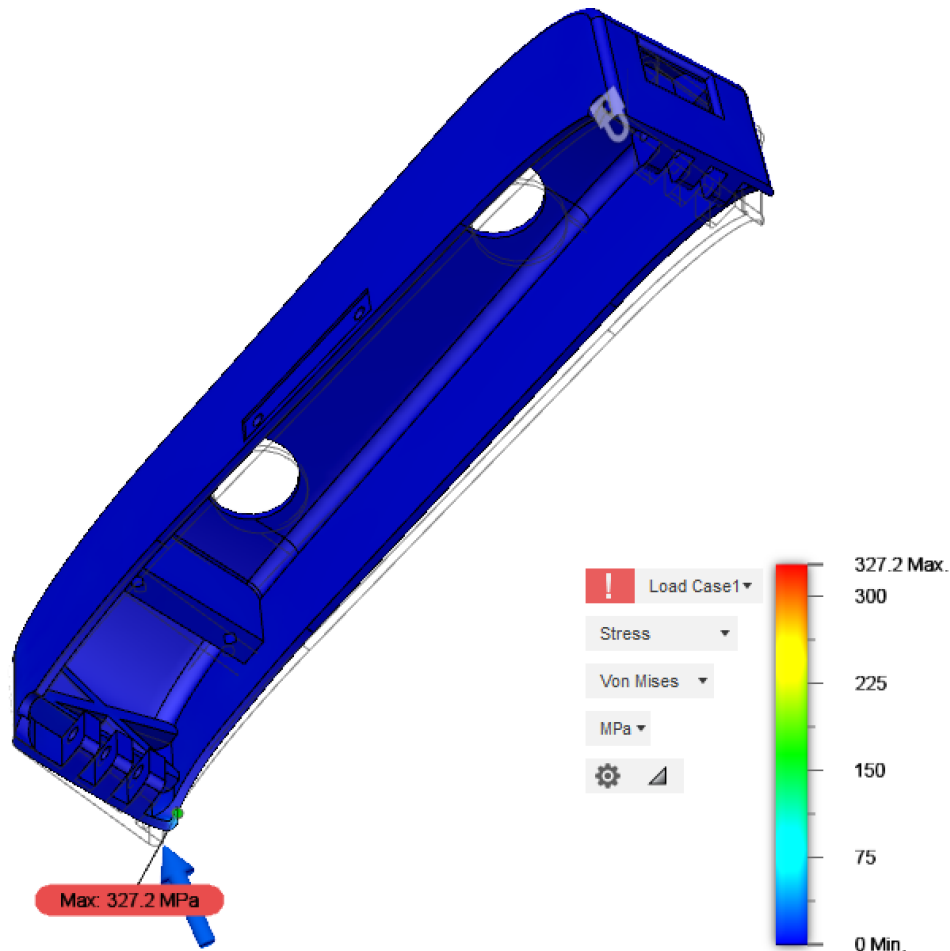


Figure C.17: *Computer simulation without the ZED mini aluminium frame mounted*

The simulation in Figure C.17 was done without the ZED Mini aluminium frame inside. We can observe a huge flex in the frame, but the damage is not plastic deformation. The frame will still return to the original shape after the damage.

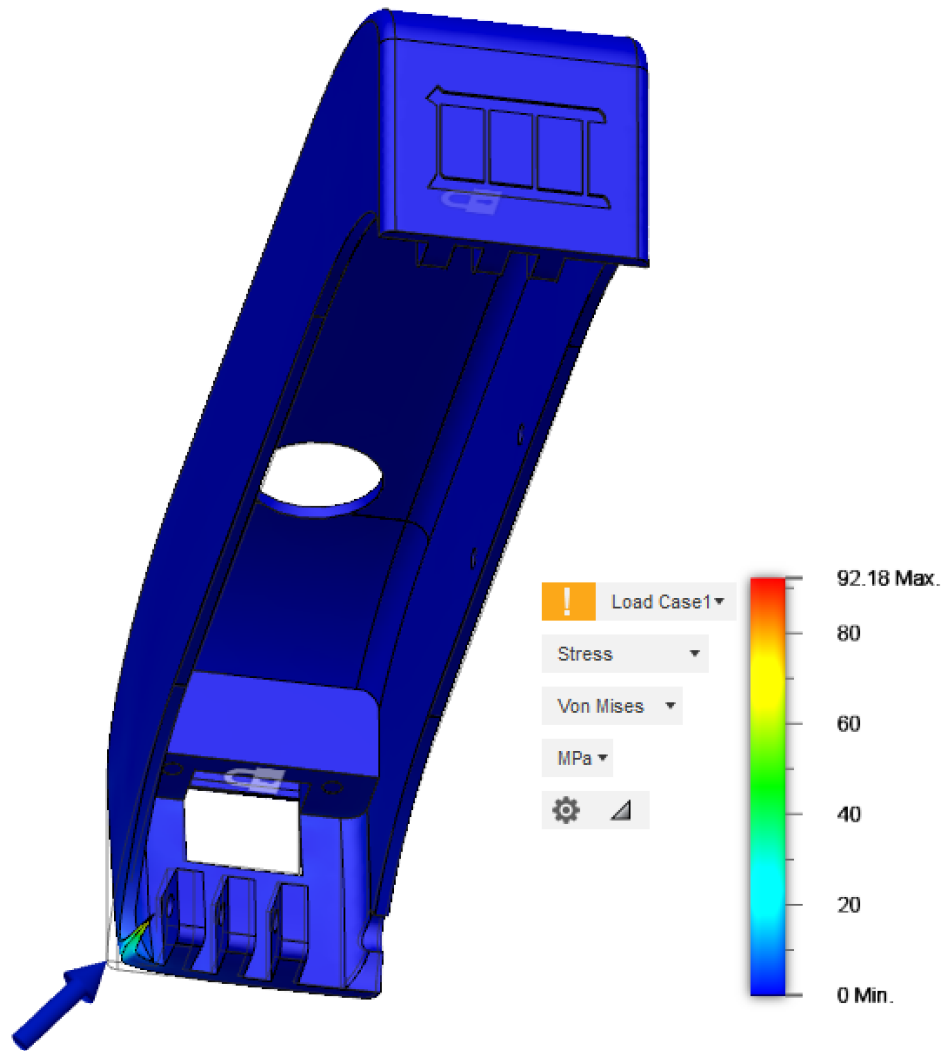


Figure C.18: *This simulation was done with the ZED Mini aluminium frame inside.*

The simulation in Figure C.18 was done with the ZED Mini aluminium frame inside. Because the housing is less free to rotate and flex during the impact, the force will be distributed through a smaller area than previously. The damage done by the impact is not permanent, as the frame should return to its original form.

D Additional attachments

In this section you can find all the additional attachments for this thesis, focusing mainly on the management side of the project.

D.1 Timetable

The timetable for the project can be seen in Figure D.1, and contains all the logged hours for each week spent on the project.

Date period	Week	Jon Petter	Jørgen	Magnus	Mohammed	Total (hours)
09.01 - 13.01	2	11,00	11,50	14,00	10,50	47,00
14.01 - 20.01	3	8,00	14,75	12,50	11,50	46,75
21.01 - 27.01	4	5,25	7,00	6,25	6,25	24,75
28.01 - 03.02	5	11,50	26,75	18,50	15,00	71,75
04.02 - 10.02	6	7,75	6,50	29,75	20,25	64,25
11.02 - 17.02	7	19,00	17,75	14,50	15,50	66,75
18.02 - 24.02	8	4,50	7,00	3,00	4,50	19,00
25.02 - 03.03	9	6,00	5,50	7,00	2,00	20,50
04.03 - 10.03	10	21,75	23,75	26,50	27,50	99,50
11.03 - 17.03	11	17,50	26,50	33,50	25,75	103,25
18.03 - 24.03	12	36,25	33,75	43,00	30,00	143,00
25.03 - 31.03	13	31,50	32,25	39,25	32,00	135,00
01.04 - 07.04	14	33,25	37,50	21,25	37,75	129,75
08.04 - 14.04	15	22,00	26,50	23,00	24,25	95,75
15.04 - 21.04	16	3,25	14,75	6,00	0,00	24,00
22.04 - 28.04	17	36,25	36,00	37,25	36,00	145,50
29.04 - 05.05	18	36,75	37,25	43,00	46,25	163,25
06.05 - 12.05	19	27,00	42,75	40,00	48,25	158,00
13.05 - 19.05	20	42,00	37,25	36,00	53,25	168,50
20.05 - 26.05	21	51,00	76,50	71,50	26,25	225,25
27.05 - 28.05	22	21,00	24,25	26,00	15,50	86,75
					TOTAL	2 038,25

Figure D.1: *The timetable for the project, with the time shown in hours and minutes, where 0,25 is 15min, 0,50 is 30min, and 0,75 is 45min.*

D.2 S-curve

The S-curve for the project can be seen in Figure D.2, and is based on the data from the timetable.

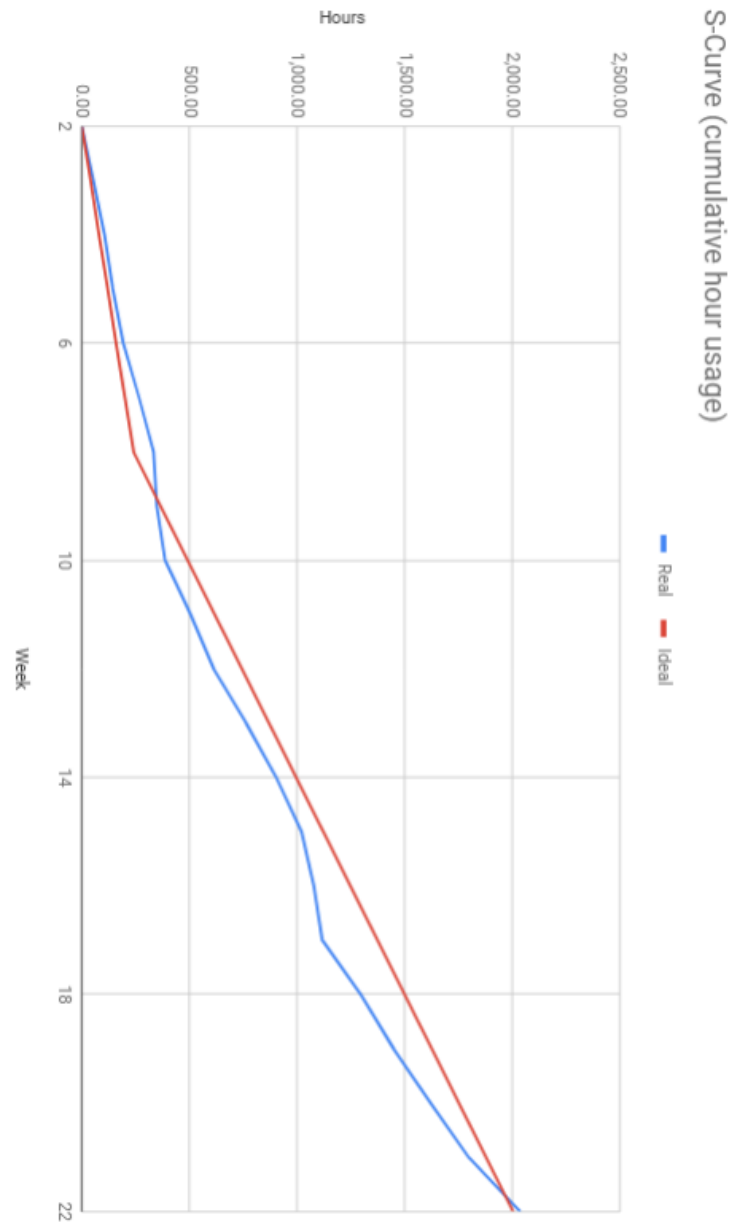


Figure D.2: *The S-curve for the project.*

D.3 Gantt diagram

The final revision of the Gantt diagram can be seen in Figure D.3.

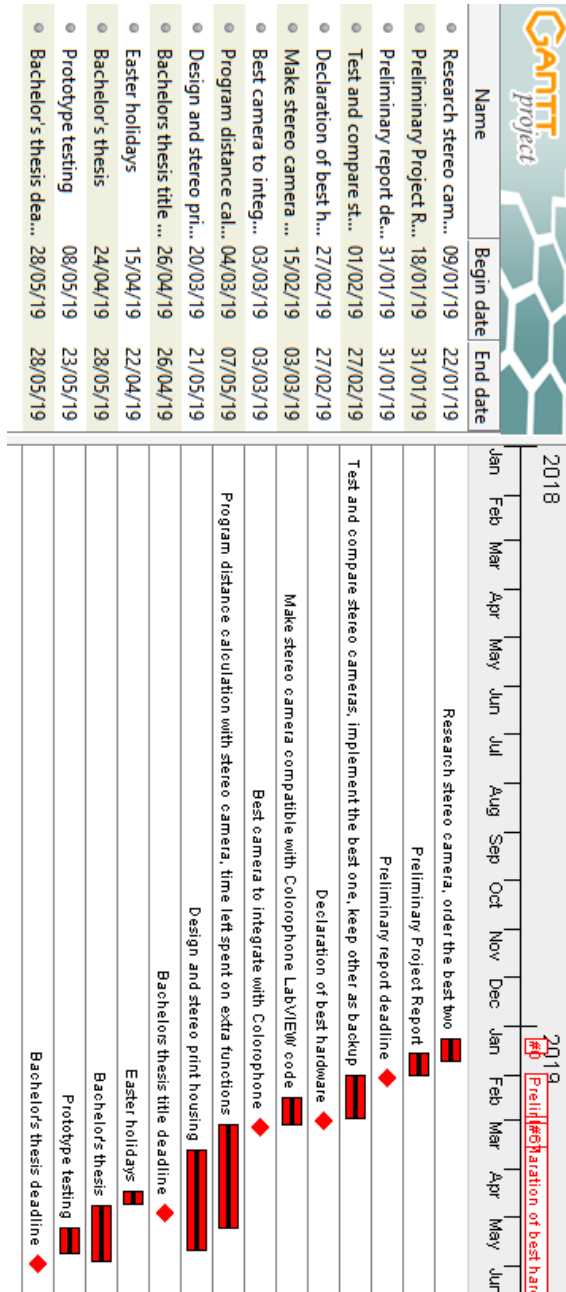


Figure D.3: *The final revision of the Gantt diagram.*

Colorophone, A Stereo Vision based Wearable Sonification System for the Visually Impaired

Colorophone is a sensory substitution device that enables visually impaired individuals to acquire visual information such as color and distance through sound. The previous versions of Colorophone used an ultrasonic sensor to detect distance [1]. In the past the Colorophone team demonstrated the potential of using eye-tracking systems. Our belief is that stereo vision in conjunction with eye tracking could improve the practical functionality of the Colorophone. Our journey started with a question: could a stereo camera expand the distance sensing abilities of the Colorophone, and replace the ultrasonic sensor as a superior alternative?

We started the project by searching for the best consumer stereo camera on the market, we had a total of eight options. We ordered the two cameras with the most promising technical specifications. The Intel D415 and Stereolabs ZED Mini utilize different types of sensors to achieve stereo vision. Our results proved that the ZED Mini was better for our use case, because it had better color and distance accuracy. Our main job was to create a prototype that could detect distance and color using a stereo camera, then retransmit the information to the user by audio.

Humans can understand how far away an object is by comparing what our left and right eyes see. Stereo cameras utilize the same concepts for estimating distances. Two camera sensors are placed on the same baseline, both perceiving the same scene from two different perspectives. Images from the sensors will depict the same object in two different positions, see Figure 1. The difference in position for the object is called binocular disparity, and is directly relatable to the distance to the object.

Digital images are a composition of pixels. Each pixel contains the value for the colour red, green and blue (RGB). By comparing the specific RGB numeric sequences from the left and right image from the stereo camera, we can find the binocular disparity. The software code starts by selecting a small area of the left image, which is then matched with a defined area in the right image. The matching is achieved by utilizing the mathematical method Sum of Squared Differences.

The color recognition algorithm calculates the mean RGB value for the pixels in the centre of the left image. The sonification system starts by creating three different sinusoidal signals with unique frequencies, for the colours red, green, and blue. The amplitude for the individual signals are dependent on the quantity of the corresponding color detected by the software. The color audio signals are added together with a ramp wave, where the frequency of the ramp wave is the inverse of the calculated distance. This ramp wave will give off a 'tick' sound, and the closer an object, the more frequent the ticks.

The final prototype is a hybrid mixture between traditional eyeglasses and augmented reality headsets. They fit on the user like eyeglasses, but we had to use a strap mechanism to achieve better weight balance. The temples are designed to be user adjustable to fit better. The nose pad is interchangeable so the user can choose the design that fits them the best. Bone conducting headphones are used so that the user still has the ability to hear their surroundings.



Figure 1, baseline is the distance between the camera lenses.

Figure 2, our 3D printed prototype.

Figure 1 taken from: <https://www.intelrealsense.com/wp-content/uploads/2018/12/stereo-sd-1.png>
 [1] D. Osinski, "A sensory substitution device inspired by the human visual system," 2018.