University of Waterloo

Department of Mechanical & Mechatronics Engineering

Capstone Design Project
Final Design Report

Group #1

Fan Zhang (20507508)
Wen Chao (Bill) Jiang (20512856)
Andy Zhouming Su (20477022)

Prepared For
Prof. Jan Huissoon
Prof. Sanjeev Bedi

April 2nd, 2018

Jan Huissoon, Professor
Department of Mechanical and Mechatronics Engineering
University of Waterloo
Waterloo, Ontario
N2L 3G1

Dear Professor Huissoon,

The following report "Capstone Design Project Final Design Report" was prepared for the MTE 482 Mechatronics Engineering Design Project course by the members of Group 1 listed below. This report was prepared during our 4B academic term. The purpose of this report is to present the final design of our project and detail our construction, commissioning, and design evaluation process, as well as results and recommendations.

We are the sole authors of this report and, unless otherwise stated and properly referenced in the report, the entire content of this report is original work done by us. We have all reviewed the report and are aware of the content. The content of this report has not received credit in this or any other course that we have taken in the past or are currently taking at this time.

Sincerely,

Fan Zhang (20507508)

Wen Chao (Bill) Jiang (20512856)

Andy Zhouming Su (20477022)

Table of Contents

List of Figures

List of Tables

Executive Summary

This report represents a culmination of the work done by Group 1 for the MTE capstone project. The need statement being addressed was "current information terminals (screens) are obstructive and distracting to use when multitasking." Our solution to this need was to design a head mounted display system, similar to Augmented Reality wearables on the market. The build cost of one unit was $266.31. The total expenditure for this project was $1008.37.

From the glasses frame design proposed in the MTE481 final report, there has been several changes since then. The compute platform became the Raspberry Pi 3 instead of the previous Raspberry Pi Zero. Due to the larger Pi, the form factor was modified to a headband wearable device. The Fresnel lens display was swapped for a color microdisplay instead.

The new design allows for modular hardware component inclusion. The components chosen for this design include a gesture pad and microphone on the right side, and a microdisplay in the center facing the left eye. These components are attached to the main frame. The Core Module, which includes the battery, Pi 3, audio amplifier, and charging circuit, is located at the back of the head. A Velcro strap kept the frame and the module in place on the user's head.

A operating system named GlassOS has been custom made to run on Pi 3. GlassOS was designed with a language agnostic handler to support any hardware drivers. GlassOS features developmental APIs to enable development of apps that can take advantage of the hardware. It also features a fluid user interface that is controlled by hand gesture or voice commands.

The prototype built has met all constraints specified for this project. Recommendations have been included to address this project's shortcomings and to suggest possible future development steps.

1.0 Introduction

1.1 Background

Modern smartphones are all-in-one computing devices that serve as our main information terminal. From them, one can obtain their messages, emails, notifications, and other personal data. Smartphones, however, require our full attention as well as direct physical interaction with our hand(s). As such, it becomes difficult, if not impossible, to use the smartphone when there is another task at hand. Nonetheless, it could be a situation where one would need the information from the smartphone. Trying to use it in such a situation could lead small inconveniences (such as bumping into another person) or even fatal mistakes (such as vehicle collisions).

There are numerous real world needs where one might need to access important data, but cannot, due to being preoccupied with a task at hand. A simple example could be a car mechanic needing access to a repair documentation, but they are holding power tools in their hands. Another example could be a delivery man who is holding a heavy package but needs to know the exact address of a shipment. In both cases, the person could've freed their hands and then accessed their smartphones, but with a better alternative method, they would not need to in the first place. There is such a need for users to obtain information from their smartphone (or other distracting information terminals), but without the interaction focus that they would typically require.

There are synergetic benefits to staying connected with the information stream from a smartphone, but it's essential to find an alternative method to obtain that information without the distracting user experience.

1.2 Needs Assessment

The need statement that this project is trying to address is "current information terminals (screens) are obstructive and distracting to use when multitasking." The smartphone is the most used device for personal computing and its screen is how one's personal data is accessed. The problem is that smartphone screens were not designed to be used simultaneously when doing another task (multitasking).

1.3 Problem Formulation

We envision a world where everyone can stay connected at all times, regardless of what activities they engage in.

Today the only products that can fulfill this need are wearable Augmented Reality (AR) platforms. Their main issue, however, is the barrier to entry arising out of their high costs. An entry level consumer AR glass costs, at the time of this report, approximately $1000 USD.

Our goal is to design a product that fulfills the very same need, but at a much lower price point.

1.4 Constraints

In an effort to engineer a proper design, it was important to impose valid constraints to ensure that the design was well scoped. These constraints were drafted to ensure that the proper design would meet the need correctly.

The first and most important constraint is that the information provided to the user must be primarily through visual means. The method used must be some kind of digital display. This display should not cover

more than 50% of the user's field of view. A coverage higher than half risks impairing the user's vision to unsafe degrees. A user should always have more than one eye worth of unaltered vision.

The second constraint states that the user should not require hand touching interaction to use the product for 95% of the time. For a typical touchscreen, this means that the user should not have to use their hands to use the device aside from an occasional touch action. If the product requires human input, other methods of interaction should be available as an alternative.

The third constraint is to impose a size and weight limit to the product. This is to ensure the device remains portable enough for regular use. The volume limit for this device is 300 mm by 300 mm by 300 mm cubic volume. The weight limit is one kilogram. A device that is bigger and heavier than these constraints is deemed to have poor ergonomics, and as such, is not suitable for regular use.

In order to assure than this device will be portable, the design will be required to be powered by a portable energy source. This does not constraint the device to be powered portably at all times, such as plugging in to charge.

The last constraint is keep the bill of material cost of the product under $1000. There two reasons for this amount. First, building a device costing over $1000 would make it more expensive than the competition and it would defeat the goal of achieving a lower price point. Second, this project receives $450 in assistance from the MME department. The team members expect that it won't be enough to bring this project to completion, so they are willing to cover any additional expenses out of pocket. They determined that another $550 is a fair amount to cover.

1.5 Criteria

In order to parameterize the design, a set of criteria was conceived. These criteria acted as key metrics to the evaluation process in order to select the best design.

The first criterion evaluates the ergonomics of the design by factoring weight, size, comfort, and view obstruction. The weight is scaled to kilograms and determined from the type and amount of material used in the design. The size is determined by the volume (centimeter cubed) in typical usage configuration. A comfort score will be determined by the team members based on their combined judgement. The view obstruction amount will be determined by estimating the percentage of field of view covered during typical use.

The second criterion is the visual aesthetic of the design. This criterion will evaluate the design in standard configuration, on a scoring system between the team members.

The third criterion is the projected cost of the design. The designs will be given an estimated quote on the cost it would take to bring the design to market. For accurate estimations, existing similar products on the market are used as weighting calibration.

The fourth criterion is the complexity of the design. This criterion will evaluate the number of man hours needed to research and develop a prototype for the capstone symposium. For a Mechatronics project, this can be divided into three areas of interest: mechanical, electrical, and software. Based on the skillset of the team members, it will affect the projected number of man hours in each area.

The last criterion is the display quality of the design. The display quality will evaluate the specifications of the digital display, such as resolution, pixel density, viewing angle, brightness. The resolution is measured

by pixel count and arrangement. The pixel density is measured by the arcminute per pixel. Viewing angle is determined by the degree range where the light intensity is higher than 50% of the maximum. The brightness is measured in nits, and in the case of this criterion, a better display should be capable of higher maximum brightness.

1.6 Design Review

In the final report from MTE 481, there were four proposed designs that could fulfill the need statement. The first the smart watch form factor: a modern watch with a small touchscreen display. The second is a Virtual Reality (VR) headset that displays a passthrough image using a camera feed. The third is a reflector headset that reflects smartphone display light on see-through glass visors. The fourth form factor is a head mounted display, which as its name implies, is a digital display that rests on the user's head. These four possible designs were evaluated based on the chosen criteria. We then used an engineering decision matrix to select the most suitable design. The chosen final design is the head mounted display form factor.

In order to minimize the build cost of the design, the decision was made to use as much open source hardware and software components as possible. Proprietary technology can be expensive to develop and manufacture.

2.0 Final Design

2.1 Final Design Details

2.1.1 Mechanical Design

Due to a UI software framework that couldn't run on the original Raspberry Pi Zero, the team has decided to change the hardware and initiate a mechanical redesign. The Figure 1 shown below shows the new mechanical design. The full mechanical drawings for each component are shown in Appendix A.

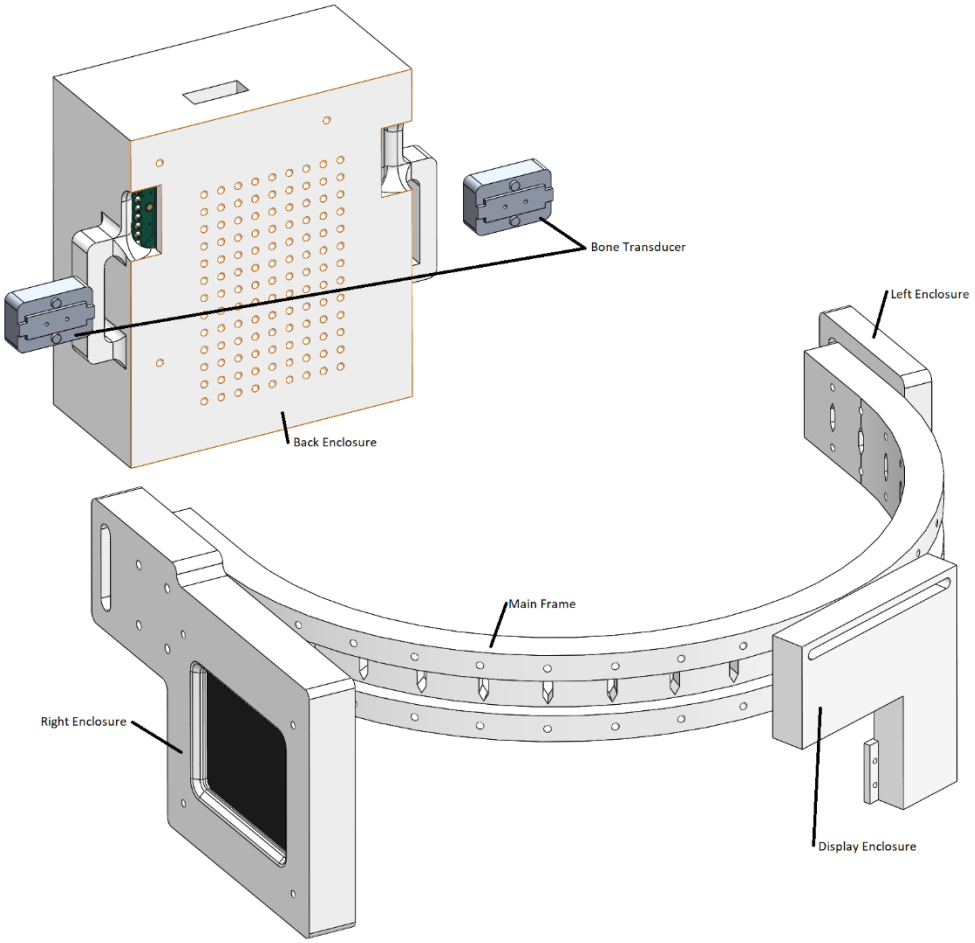


Figure 1: Model of the head mounted display excluding elastic strap, display, wires, screws, glue, and tape.

2.1.1.1 Modular Design

To speed up the pace of development, the new mechanical design focuses on a modular design. There is a main frame, which sits on the user's forehead, that has many mount holes. The 3 components that connect to the main frame is the left and right enclosure and the display enclosure. The back of the head set is a box enclosure that contains many electronics. The advantage of this design method is that once the main frame is design, it can be printed while designing other components. If any single enclosure needs to be redesign, other components are unaffected. This method saves time and effort since deadline is tight and the redesign was started in mid-February.

2.1.1.2 Component Assembly

Almost all the components that links together are using standard M2.5 screws. This screw is chosen because it is the standard mounting screw for many of the breakout boards and the Raspberry Pi. The left, right, and display enclosures are mounted onto the main frame using M2.5 screws. The main frame has holes tapped for M2.5 screws. The left and right enclosure also have a large slit that can fit an elastic strap. There is a matching set of slit on the back enclosure. There is a single elastic strap, with a Velcro portion sewn onto it, that connects to the front assembly and the back enclosure. The Velcro allows for adjustment of the strap. The bone conduction speaker is also attached onto the strap using hot glue for direct conduction onto the user's head.

2.1.1.3 Display Enclosure

The display enclosure, also known as Display Module, is designed to mount the display assembly and the display adapter board. The adapter board was held onto the enclosure using double sided tape due to the non-standard hole size. The enclosure also sports a set of mounting holes for the actual display to mount on, using the screws that came with the display. Instead holes, the display enclosure features a large slit for

mounting onto the main frame using M2.5 screws. The reason for this is that the slit allows for fine adjustment of the display location so that it can be fit for each user.

2.1.1.4 Left and Right Enclosure

As mentioned before, the left and right enclosure contain holes for mount onto the main frame as well as slit for elastic Velcro strap. The left enclosure contains no components and is purely used for the strap. The right enclosure, also known as the Control Module, contains a central cutout for the gesture pad as well as mounting holes for the gesture pad. In addition, the right enclosure also contains the microphone. Finally, there is a transparent Plexiglas cover that holds everything together. Figure 2 shows all the components and their location.

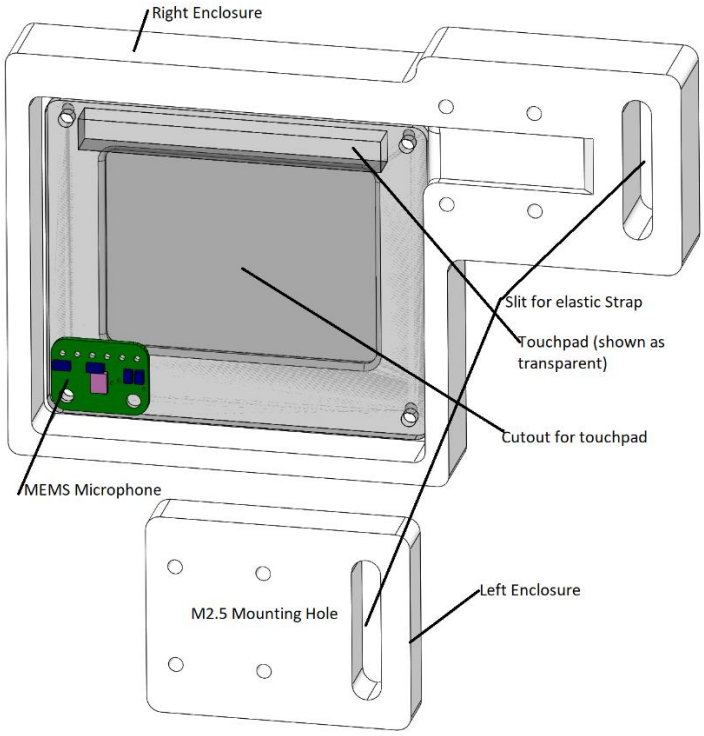


Figure 2: Right and left enclosure with touch pad shown as transparent and without Plexiglas Cover.

2.1.1.5 Back Enclosure

The back enclosure, also known as the Core Module, contains many of the important electronic components for the head mounted display: the processing unit (Raspberry Pi 3), audio amplifier, and the battery charging circuit. Figure 3 shows the components inside the enclosure.

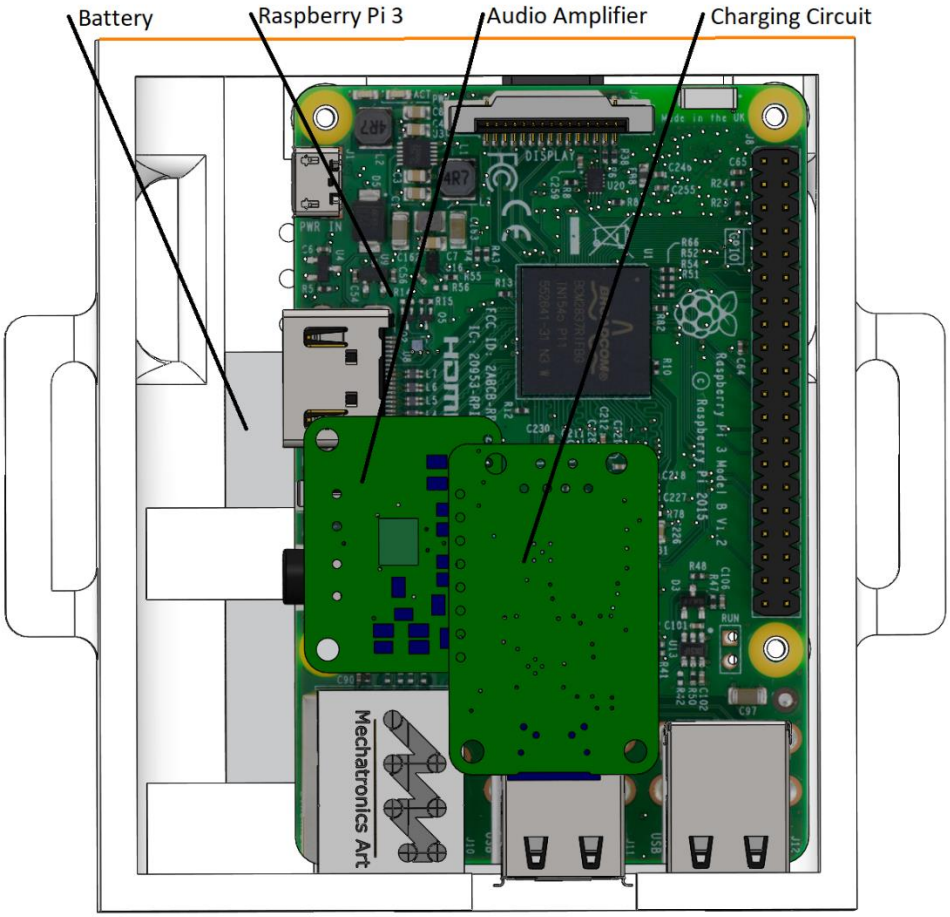


Figure 3: Inside the back enclosure without Plexiglas cover, screws, and wires.

The back enclosure also features a set of 25 mm by 5 mm slit for the elastic strap, like the right and left enclosure. Figure 4 shows an exploded view of all the components in the back enclosure.

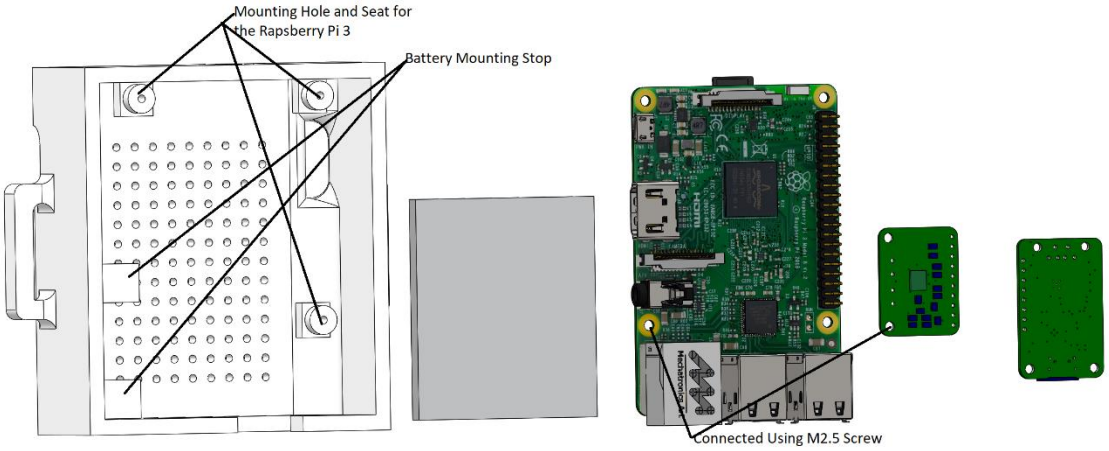


Figure 4: Exploded view of the back enclosure excluding screws and Plexiglas cover.

The battery is mounted onto the bottom of the enclosure using a set of mounting stops and tape. Then the Pi is subsequently mounted onto the 3 M2.5 mounting hole and seats. The audio amplifier is then mounted onto the Pi's remaining hole. Lastly, the charging circuit is mounted onto the Plexiglas cover that sits on top of the enclosure. The cover is held in place using long M2.5 screws.

2.1.1.6 Finite Element Analysis

The major components of the headset were also analyzed using Finite Element Analysis tools provided by SolidWorks. The components were able to survive a 1.8-meter drop test with deformations in the order of 10^{-4} meters which would not be permanent. The results of these tests are shown in Appendix B. In addition, the major components were also subject to a static loading test with a factor of safety of 2.0 on anticipated external force acting on each component. In this case the component exhibited even smaller deformations in the order of 10^{-6} meters which will not result in permanent deformation. The results of these tests are shown in Appendix C. It can be concluded that the components are overdesigned for anticipated loads and are not at risk of failure.

2.1.2 Hardware Features

The device is computationally driven by a Raspberry Pi 3 Model B, located in the Core Module. The Raspberry Pi also provides connectivity through WiFi and Bluetooth. It is equipped with a camera viewfinder style color microdisplay to display the user interface and serve information through visual means. The user can interact with the device through hand gestures on top of the gesture sensor located in the Control Module. In addition, the user can also interact with the device through voice commands that are registered through the built-in microphone, also located in the Control Module. The device also includes built in speakers in the form of bone conduction transducers mounted at the back of the Core Module that is able to deliver information to the user through auditory means as well as to play music. To power the device, an integrated battery provides up to 4 hours of theoretical run time (tested to last over 3 hours in real world use scenarios). The battery can be recharged through a microUSB port exposed in the Core Module.

2.1.3 User Interface

The user interface and interactions were designed around the microdisplay with a high tolerance for user error. Given that the purpose of the device was to minimize distractions for the user, the interface presents small amounts of at-a-glance information at a time. Large font size and iconography are used to ensure readability, as shown in example views of the home screen in Figure 5.

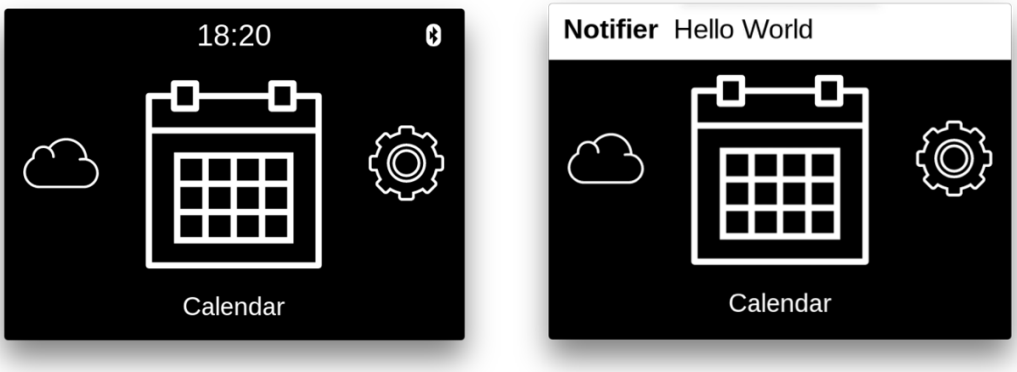


Figure 5: Views of the home screen (left) and with a sample notification popup (right).

Users can interact with the device through the user of gestures. Although the gesture sensor provides such precision that made it capable of precise mouse movements, operating a head mounted trackpad is both distracting and prone to error. Instead, only swipe and tap gestures are used to operate the interface. On the home screen, the user is able to swipe left and right to browse available apps, and open one with a tap. A downwards swipe from the home screen opens the notifications list as shown in Figure 6, where again the user can swipe left and right to go through available notifications. Globally, an upwards swipe closes the open app and returns to the home screen. The user can also use voice command by starting a query with "Ok Google". A Google Assistant specific UI indicates that the device is listening for a query and will display the user command as text once processed.

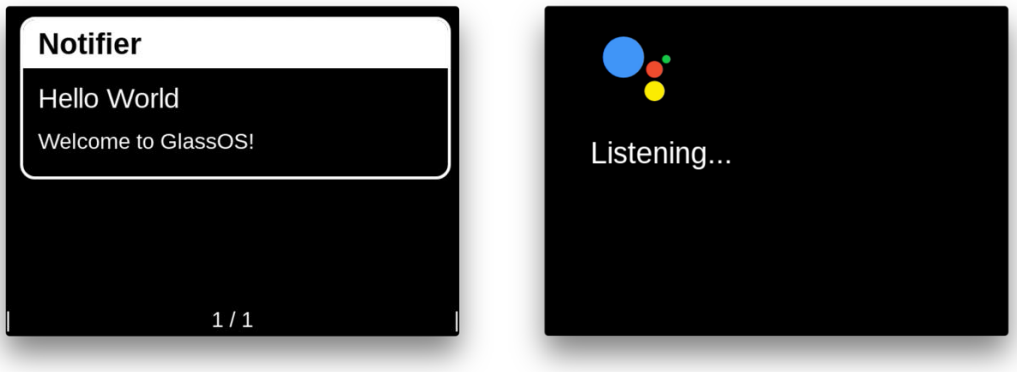


Figure 6: A view of the notifications list (left) and the Google Assistant prompt (right).

2.1.4 Software Features

The device operating system, dubbed GlassOS, includes a core set of features. The device is able to communicate with Android smartphones through a companion app, which relays phone notifications to the device through Bluetooth. Google Assistant is deeply integrated into the operating system to accept voice commands. Users can request information, such as the current weather, through the smart assistant. In addition, users can also open and control applications within the system through the smart assistant.

Above all, the key feature of GlassOS is the ability to easily create additional applications. The software architecture design is heavily abstracted so that device hardware can easily be utilized through a simple application programming interface (API). Applications can access backends and external APIs through the built in WiFi. Some sample applications which demonstrate these capability, including Weather, Calendar, and Music Player, are shown in Figure 7.

Figure 7: Sample weather, calendar, and music player applications.

2.2 Modifications & Deviations from Original Design

The final design deviated largely from the original design in order to achieve a more optimal software experience. The original design described a form factor similar to a pair of glasses. This was largely made possible by the small original compute platform Raspberry Pi Zero, which was discovered to be insufficient

for achieving an optimal software experience. This section explains, in chronological order, the series of decisions and changes which led to the final design.

2.2.1 Compute Platform

Central to the operations of the device is the compute platform. Several solutions were initially explored, including popular Arduino microcontrollers (Arduino Nano, Arduino MKR Series, Arduino Yún Mini), as well as the Raspberry Pi line of system-on-a-chip (SoC) platforms (Pi 3 Model B, Pi Zero). The Raspberry Pi Zero was initially preferred for its small form factor, low power consumption, and substantial computing power. However, through testing it was discovered that its ARMv6 single-core processor architecture was incompatible and insufficient to run an optimal software stack. The SoC struggled especially in graphically intensive tasks, to the point where the user interface of the default operating system Raspbian lagged under normal use. Continuing the design with the Raspberry Pi Zero would have led to an optimal form factor but a subpar software experience.

For these reasons, several more powerful alternatives were explored, including the Raspberry Pi 3 Model B and its commercially available derivatives. The Pi 3 Model B was an acceptable baseline alternative as it embeds a more powerful and modern quad-core ARM Cortex-A53, along with being readily available and widely supported. Derivative platforms such as the Banana Pi Zero and NanoPi-M3, which incorporate similarly powerful processors and onboard features at a smaller form factor, were also considered. Despite being superior to the Pi 3 Model B in specifications, these SoC solutions were ultimately not used as they lacked availability and community support. Ultimately, the Raspberry Pi 3 Model B was chosen as the next most viable compute platform.

2.2.2 Form Factor

Due to the change in hardware, it was no longer possible to mount the electrical components in the previous form factor. Instead, the new form factor is akin to a head band. The touch pad and the microphone are place on the right side for easy right-hand access and mouth access. The other electrical components are placed in the back, where there is more room and does not obstruct the user.

2.2.3 Display

The final display used for the project is one we acquired from a chinese manufacturer Banggood. It is a 320 pixel by 240 pixel color microdisplay. **Figure 8** shows the display.

Figure 8: The microdisplay used for this project.

The reason this display is chosen is a combination of its low price (CA $79.14 at the time of writing) and easy interfacing with the Raspberry Pi (analog video header). The full specification of the display is list below in Figure 9.

Description:
Weight: 1g
Dimensions: 15.2x11.3x9.2mm
Resolution: QVGA 320X240 full color.
Low Power Consumption: <100mW
Display Panel Diagonal: 5mm(0.20")
Display Pixel Size: 13.5um
Pixel Fill Rate: 94%
Color Depth: 4200000 color
Display Frame Rate: 120HZ(NTSC),100HZ(PAL)
Contrast Ratio: 80:1
Operating Voltage: 3.7-5V
Working Temperature: -10℃-70℃
The effective area of the display panel: 4.05x3.02mm
Supported Formats :8-bit RGB-serial data,Hd,Vd,Valid,Clock CCIR 601,CCIR656
FLCOS ferroelectric liquid crystal micro-display technology manufacturing.

Figure 9: The display specification.

2.2.4 Touch Interface

The original design proposed to use a custom capacitive touch solution based on the Adafruit MPR121 Touch Sensor. The touch sensor supports up to twelve touch zone which could be created out of any conductive material. As the sensor was merely a breakout board and did not include the actual touch surface, plans were made to construct the touch surface using isolated pieces of metallic foil arranged in a 3 by 4 grid on top of a non-conductive material such as Plexiglas or ABS plastic. An illustration of this configuration is shown in Figure 10.

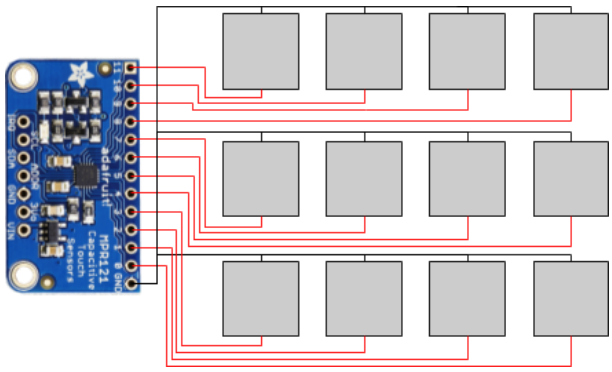


Figure 10: Proposed touchpad configuration with the Adafruit MPR 121 Touch Sensor.

Construction of such a touchpad is difficult and may not result in a robust end product. In addition, it is difficult to miniaturize such a trackpad without perhaps commissioning a custom PCB. These difficulties led to a search for alternatives and the eventually discovery of the Pimoroni Skywriter HAT. The Skywriter is able to discretize up to 150 positions per inch [1] (resolution similar to a typical notebook trackpad) and was designed to interface with Raspberry Pi devices. Being superior in every aspect (with exception of price) to the original solution, the Skywriter was adopted as the new touch interface for the final design.

2.3 Construction & Development

2.3.1 3D Printing

Majority of the mechanical components used in this project are 3D printed using the Cubicon Single+ 3D printer. These printers can be accessed cheaply in the WATIMAKE maker space and for free at Kitchener Public Library. Although previous prototype versions are printed using faster and lower resolution settings, however, the parts used in the final prototype were printed used highest settings. Figure 11 shown below detailed the print setting used.

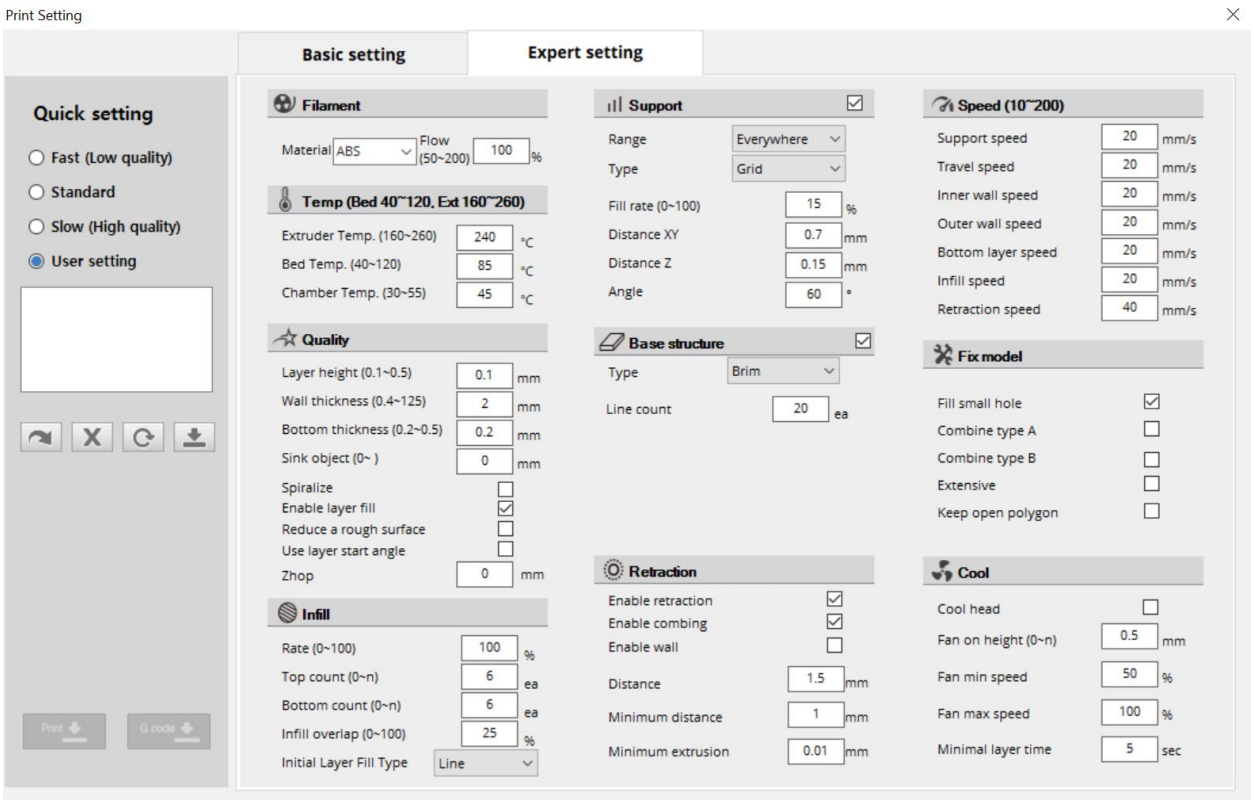


Figure 11: 3D printing setting used for the project.

All the parts used ABS as the material of choice. In addition, the parts were post processed with filer to smooth out the surfaces, drills to enlarge printed hole into the correct dimension, and tap drills to thread the holes.

2.3.2 Assembly

As mentioned in the mechanical section previously, the bulk of the parts are assembly together using M2.5 screws of various length. The front and back section of the head set are held together using an elastic strap. The strap has Velcro sewn on top of it for fine adjustment. There is also use of electrical tape that act as

insulation and held some small, non-structural component in place. Lastly, the bone conduction speakers are secured onto the elastic strap using hot glue.

2.3.3 Electrical Assembly

The final design makes use of mostly off-the-shelf components to stay in line with the project goal of being open source friendly. Commercially available parts are generally more reliable as compared to custom solutions. This section details the components used and their interconnections. A full wiring diagram can be found in Appendix D.

2.3.3.1 Compute Platform

The most important component of this prototype device is the compute platform, which is a Raspberry Pi 3 Model B. The credit card sized board embeds a quad-core 1.2 GHz ARM Cortex-A53 system-on-a-chip (SoC) processor, 1 GB onboard memory, and both 802.11n wireless networking and Bluetooth capabilities [2]. The Peripheral components can be interfaced with the platform through its 40 general-purpose-input/output (GPIO) pins which support a number of common communication protocols including Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I^2C), Inter-IC Sound (I^2S), and Pulse Width Modulation (PWM). The platform also outputs analog audio and video signals through a standard 3.5mm phone connector.

The peripherals to this central computing platform can be divided into three categories: analog, digital, and power. The digital components were connected through the GPIO pins and communicate using the I^2C bus. The analog components can be connected via any of the PWM-capable GPIO pins or through the 3.5mm phone connector. For the sake of reducing the number of connections and consolidating wiring, analog components exclusively used the 3.5mm phone connector. The power components supply power to the

Raspberry Pi (which in turns power all other peripherals through its 3.3V and 5V rails) through a microUSB port to utilize the onboard power protection circuit.

2.3.3.2 Digital Components

The digital components of the system include the touch/gesture input device and the audio input device. The Pimoroni Skywriter HAT is used for capturing user touch and gesture interactions with the system. This component, shown in Figure 12, is a PCB board not much larger than a credit card and uses electromagnetic near-field deviations to detect not only touches but also gestures up to 5 centimeters above the surface [1]. This device was designed as an extension board or HAT to be attached on top of the Raspberry Pi, hence the 40 pin female headers.

Figure 12: Pimoroni Skywriter HAT gesture sensor.

However, it was not ideal to directly attached the sensor to the Raspberry Pi as it would occupy all 40 GPIO pins and constrain the placement of the touch surface. Instead, the sensor is connected through the bare minimum 6 GPIO pins necessary for its operation. The pins include 3.3V power, ground, and I^2C related pins (SDA, SCL, Transfer, and Reset).

For audio input, the Adafruit I^{2}S MEMS microphone was used. This solid-state microphone is small and able to communicate with the Raspberry Pi digitally using I^{2}S (a serial bus interface similar to I^{2}C but for audio data). It requires a total of 5 GPIO pins: 2 pins for power and ground, 2 pins for data transfer and synchronization (DOUT and BCLK), and 1 pin to set left or right channel (LRCL). Optionally, two of such microphones can be used in conjunction with the device select pin (SEL) to achieve stereo input, but this was not done as it is unnecessary to the purpose of the microphone in this system.

The Raspberry Pi's GPIO pins are almost exclusively used by digital components, as shown in Figure 13. A total of 15 pins are occupied, 11 of which includes corresponds to the digital components described in this section. An additional 4 pins (two 5V power and two grounds) are used to supply power to the display and audio amplifier. Although all power and serial bus related pins are occupied, they can be shared between components in the case of future expansion.

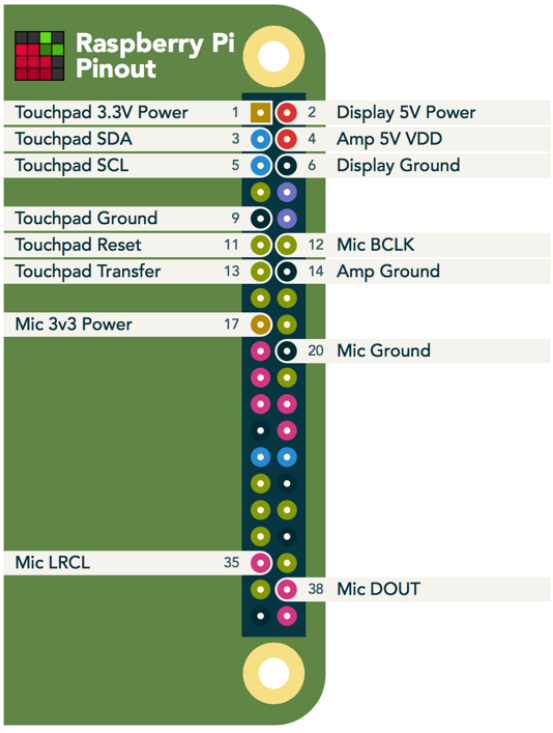


Figure 13: Pinout diagram for the Raspberry Pi's 40 GPIO pins. Not all pins are used.

2.3.3.3 Analog Components

The analog components of the system include audio and video output. Modern models of the Raspberry Pi, including the Pi 3 Model B used in this design, provide a consolidated interface for analog multimedia signals through a 4-segment 3.5mm phone connector. Different devices use slightly different signaling standards even with this same connector, as shown in Figure 14, which made suitable off-the-shelf cables difficult to find. A custom connector was created using a purchased unpopulated 3.5mm header. This allowed for flexibility in the length and gauge of the wiring and reduces the number of possible discontinuities between the connector and the end device. The specific right-angle connector used also conserves space inside the Core Module.

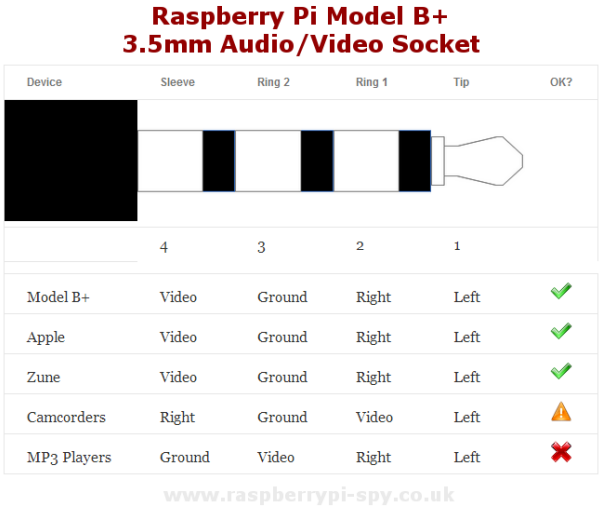


Figure 14: Segment signal standards of different devices for 4-segment 3.5mm phone connectors.

The display used is a full color Ferroelectric Liquid Crystal on Silicon (FLCOS) microdisplay which measures only 5 millimeters diagonally. The display is able to achieve a much smaller size compared to traditional Liquid Crystal Display (LCD) and Light-Emitting Diode (LED) displays due to the single die manufacturing process made possible by ferroelectric liquid crystals [3]. The display area is magnified by

a biconvex lens to approximately 20 centimeters (7.87 inches) diagonally when viewed from a focal length around 10 millimeters in front of the eye. The particular FLCOS microdisplay used in this design has a resolution of 320 pixels by 240 pixels. Although higher resolution variants exist, there is little marginal benefit given the amount of information that is intended to be displayed.

The microdisplay interfaces through a small adapter breakout board provided by the manufacturer. The operations of the display require four pins: two for power and ground, and two for analog video and analog ground, as shown in Figure 15. The two power pins source power from the GPIO pins as explained in the previous section. The analog signal pins connect to the sleeve and lower ring of the 3.5mm phone connector which provide video signal and analog ground, respectively. In addition to converting analog video signal, the adapter also allows digital control of backlight intensity through the top three pins. Brightness control was not implemented as the backlight should be kept at the default (maximum) intensity to maximize visibility.

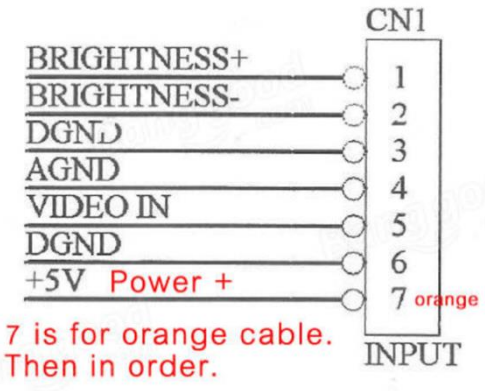


Figure 15: Pinout diagram for the display adapter provided by the manufacturer.

Audio output is produced by a pair of bone conduction transducers. These solid-state components utilize the properties of metallic materials in a magnetic field to create vibrations [4]. The main difference between these transducers and typical speakers is the lack of a diaphragm which produces audible waves. In this

particular use case, the vibrations are conducted through the skull or jawbones and interpreted by the eardrum as sound. Since the audibility of this device is does not depend on air as a medium and does not obstruct the ear canal, it does not impede the listener from hearing the environment around them. Conversely, the listener can still hear this device in a loud environment by plugging their ears (eliminates sound which travels through the air).

Electrically, the transducers are simple as the magnetic field can be directly controlled through an alternating signal. Each transducer has a signal and ground wire which can be directly connected to the left, right, and ground terminals of the 3.5mm phone connector. However, given that these transducers must produce much strong vibrations than diaphragm speaker actuators, the Raspberry Pi's connector power output is insufficient. These particular transducers have an impedance of 8 Ohms [5], which is characteristic of bookshelf size speakers. To mitigate this issue, the Adafruit MAX98306 Stereo Audio Amplifier was used as an intermediate signal amplifier. By drawing power from the Raspberry Pi 5 Volt GPIO pins, the amplifier is capable of boosting transducer inputs to 1.7 Watts per channel [6], which is appropriately high but within the transducer's 2 Watt power limit [5].

2.3.3.4 Power Components

To achieve a cordless experience, the components of the system must be powered by an integrated portable energy source. Lithium Polymer (LiPo) batteries are ideal for this system as it is more compact in size than common Lithium Ion batteries and readily available in a variety of capacities and dimensions [7]. Given the form factor of the final design does not pose a heavy constraint on the physical size of the battery, selection of the battery was based on desired runtime. Table 1 shows the current draw of components in the system. To achieve approximately four hours of runtime under typical use, a battery capacity of approximately 2000 mAh was chosen. The chosen LiPo battery has dimensions of 60 millimeters tall by 36 millimeters wide to match closely with the dimensions of the Raspberry Pi so that they can be stack within

any housing. The battery does not add significant thickness or weight to the system at only 7 millimeters thick and 34 grams in weight.

Table 1: Power draw of all components in the system.

Component	Power Draw			Assumptions
	Idle	Max	Typical	
Raspberry Pi 3 Model B [8]	260 mA	480 mA	425 mA	Average 75% Load
Speakers & Amplifier [6]	10 µA	2 mA	2 mA	Always On
Display [9]	20 mA	20 mA	20 mA	Always On
Gesture Sensor [1]	110 µA	20 mA	20 mA	Always On
Microphone [10]	75 µA	110 µA	110 µA	Always On
Total	**~280 mA**	**~522 mA**	**~467 mA**	

Typical single cell LiPo batteries have a nominal voltage of 3.7 V which is insufficient to power the 5V Raspberry Pi. In addition, properly charging a LiPo battery requires expertise or a dedicated charger. For these reasons, an Adafruit PowerBoost 1000 charger was added to the system. With this charger, this device can be charged with a microUSB cable like most modern consumer electronics. The charger is able to charge the battery at a rate up to 1000 mA while simultaneously powering the Raspberry Pi [11]. This charger incorporates a buck converter to boost the 3.7 VDC output from the LiPo battery to 5.2 VDC while allowing a current draw of up to 2 Amperes [11]. Although the Raspberry Pi may require up to 2.5 Amperes of current for spikes in performance demand, the 2 Amperes supply current prevents throttling in most situations.

In this device, the charger acts as a hub for all power in the device. The battery leads connect to the charger to provide portable power. An external power source can be connected to recharge the battery via the microUSB port. The charger provides power through its full-sized USB port which connects to the Raspberry Pi's power microUSB port. LEDs on the charger provide indication of battery level and are visible through the clear plexiglass of the housing. A switch between the enable (EN) and ground (GND)

pins was added to act as a master switch to power the system on and off. Figure 16 shows the position of the master power switch and visibility of the status indicator LEDs.

Figure 16: Overhead view of the Core Module with the master power switch and status indicator LED.

2.3.4 Wiring

To minimize bulk and impact on aesthetics from excess wire lengths, as well as to reduce possible points of discontinuity, wires were custom made to connect each component directly to the central computing platform. The length of each wire was determined by measuring from the anticipated location of the component to the connecting terminal in the Core Module with the adjustable Velcro headband extended to maximum length. The 24 AWG copper wires used are insulated with a black polyethylene sleeve along its length. To ensure mating compatibility with the male Amphenol Mini-PV headers used by the Raspberry Pi GPIO pins, all wires are terminated where necessary with Mini-PV headers.

Three special cases were made for wiring termination in this design. Firstly, the analog components which connect to the Raspberry Pi through the 3.5mm phone connector had their wires soldered directly to the

3.5mm header. Secondly, to conserve vertical space in the Control Module above female headers of the gesture sensor, wires were terminated using a Mini-PV header without a plastic sleeve so they could be bent and connected at 90 degrees. Heat shrink tubing was used at the ends to insulate as much of the exposed metal header as possible to prevent short circuiting. Lastly, a short custom USB to right angle microUSB cable was made to connect the PowerBoost 1000 charging board to the Raspberry Pi as the space inside the Core Module was severely constrained. The 90-degree connectors and custom USB cable are shown in Figure 17.

Figure 17: Wiring inside the Control Module with 90-degree male connectors (left) and custom USB to microUSB cable (right).

2.3.5 Software Development

The architectural design of the "operating system" of this device, dubbed GlassOS, took inspiration from common backend abstraction paradigms as well as the original iPhone OS. Figure 18 shows a high-level overview of the system. Many web and mobile application stacks uses a Model-View-Controller (MVC)

architecture, where a data model is converted into a view which the user can use to interact with the system via the controller. Typically, the data model comes from a database or persistent storage, but within this context, data also comes from hardware and their drivers. To bridge communication with hardware drivers and translate their data format into a consistent internal representation, an intermediate handler layer was introduced. This section will go into detail on functions and implementations of each layer in the system.

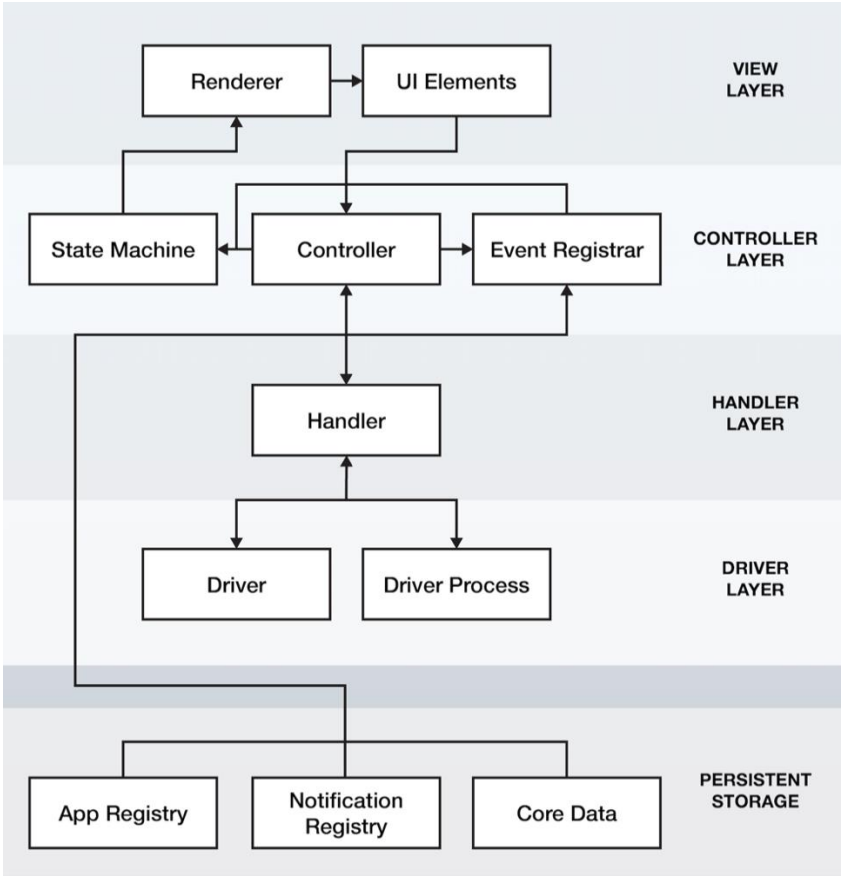


Figure 18: High level overview of the software architecture of GlassOS.

2.3.5.1 Language & Frameworks

The codebase was mostly written in JavaScript to utilize several open source frameworks which reduce overhead of implementation. Parts of the view and controller layer was built on top of popular web frontend

framework React developed by Facebook. React provides an efficient renderer which will automatically re-render the interface to reflect changes in the state machine. The user interface developed through React uses web technologies which is both well documented and flexible for implementing UI.

To bridge the gap in communication between higher level web technologies and lower level system functions as well as hardware, Node.js and the Electron framework was used. Node.js is a JavaScript runtime that enables access to kernel functions such as accessing the underlying file system and the ability to spawn and kill processes. On its own, Node.js is incapable of communicating with a React frontend without using some inter-process communication protocol such as HTTP. To improve efficiency, Electron was used. Electron is a desktop application framework developed by Github which merges a Node.js runtime environment with a sandbox for frontend code. This enables the entire stack to run within the same process and share memory space for minimal latency access.

The ability of Node.js to spawn child processes from executable and binaries enables freedom of language and frameworks for hardware interfacing. Manufacturer provided or custom drivers of any language can communicate with the system as long as a translation is defined within the handler layer. Most drivers used in this design were written in Python.

2.3.5.2 Driver Layer

The driver layer consists of language agnostic code which interfaces with the hardware. Drivers can be generalized into one of two categories: drivers and driver processes. The key difference between driver and driver processes is in how the handler can use it to access data from hardware. The prior is for synchronous and causal actions such as polling the current value, whereas the latter is for asynchronous actions such as waiting and reacting on Bluetooth events. Some components, such as the display and audio output, are driven by existing modules embedded in the Linux kernel which will not be discussed in report.

The system makes use of hardware driver processes for the gesture sensor and Bluetooth written in Python. The gesture sensor driver process uses a Python library provided by the manufacturer Pimoroni to detect gesture events and pipe the appropriate data into standard output to be read by the associated handler. The Bluetooth driver process uses a Python library for interfacing with Bluetooth Low Energy (BLE) devices using Generic Attribute Profile (GATT) protocol to alert the system of new data as well as connection and disconnection events. Both drivers are spawned as child processes by their corresponding handlers at the start of the main process.

In addition, a third driver process exists which leverages the microphone. This driver process is slight different in that it also handles smart assistant related functionality intermediately. The sole purpose of the microphone in this system is to listen for smart assistant queries. Since the Google Assistant library exists in Python, the driver process for the microphone was combined with the smart assistant related code so that the main process did not have to allocate processing to redirect audio data from the microphone to an assistant module. The driver process directly pipes events and voice queries as text to the associated handler.

2.3.5.3 Handler Layer

The handler layer acts as a translator between the language agnostic drivers in the driver layer below and the JavaScript code in layers above. For every driver there is an associated handler. Consistent in all handlers are three main functions. Firstly, the handler triggers the execution of a driver or spawns a child process for a driver process. Then, the handler initiates a stream reader for the driver or child process standard output stream. Finally, the handler registers listeners and callbacks for standard output events such as opening, receiving new data, and closing.

Handlers makes use of POSIX standard streams, specifically standard output (stdout) and standard error (stderr), to receive data from drivers running in child processes. Different messages or payloads are

delimited by a newline within the stream. Since the child process does not output to a terminal (not TTY), by default the buffer synchronization behavior is unpredictable [12]. For this reason, all drivers are written to flush stream buffers after every newline to ensure synchronization of buffer content. Although this is not the most sophisticated Remote Procedure Call (RPC) messaging protocol, it was sufficient for the low volume of time-insensitive payloads from driver processes.

Given that the majority of the system, including handlers, runs on JavaScript, the internal representation of data is JavaScript objects. Thus, for consistency, drivers were programmed to format payloads intended for handlers in JavaScript Object Notation (JSON). This allows handlers to directly parse driver payloads into native objects for further processing. This choice in notation did not impact the language agnostic nature of the driver layer seeing as most languages have support libraries for encoding and decoding JSON.

In addition to listening for driver events, each handler also emits events through an event emitter (Node.js EventEmitter) created on startup. When a handler receives an event or payload of interest from its driver process, it will in turn broadcast an event to any registered listeners of the handler's event emitter. The event may be broadcasted without a payload for signaling purposes or include a transformed version of the payload received from the driver process.

2.3.5.4 Controller Layer

The controller layer contains the business logic of the system. Following the modularity paradigm of the React framework, every application or view in the system is a component with its own state machine, controller, and event registrar, which combine to define its behavior. An example view shown in Figure 19 will be used to illustrate how these parts work together to form an interactable user interface.

Figure 19: An example view of the user interface.

This particular view of the home screen contains two components. One component is the status bar at the top of the screen which shows the time, status indicators such as Bluetooth connectivity, and phone notifications when relevant. The component's state machine maintains states such as current time and Bluetooth status. The component contains a timer which prompts an update to the time state every 60 seconds, which causes the renderer to re-render the time displayed accordingly. In order to stay up-to-date on the current Bluetooth status, the component registers a listener for Bluetooth connection and disconnection events in the event registrar. When such a disconnect event is broadcasted by the Bluetooth handler, a callback is triggered which updates the Bluetooth state in the component and causes a renderer to remove the Bluetooth symbol from the status bar. The status bar also responds to user gestures by registering listeners for swipe events. When a downward swipe event is broadcasted, the component controller opens the notifications drawer component to display notifications instead.

In essence, the state machine dictates what the user sees. The event registrar allows mutation of the state by other parts of the system. The controller allows mutation of the state in the component or other parts of the system by user interactions. With this implementation, GlassOS has to flexibility to support applications of any functionality the developer desires.

2.3.5.5 View Layer

The view layer is largely facilitated by the React framework's renderer and templating functionalities. The user interface can be defined using web markup language HTML and styled using CSS as a template with variables in place of dynamic content. During rendering, the variables are replaced with values from the state machine to produce the resulting visuals. Revisiting the example view shown in Figure 19, the status bar is templated such that the time is displayed in the middle and small icons to the right. When rendered, the current time state is inserted into the position defined in the template. Conditionals can be used in the template to determine if something should be rendered based on the current state. An example would be only rendering the Bluetooth icon if a Bluetooth device is connected.

2.3.5.6 Persistent Storage

Persistent storage is necessary in the system to preserve user data and configurations even if the device was turned off. The system relies on two configuration stores in the form of JSON files. The app registry contains the applications installed and their metadata such as name and icon. Theoretically, if a third-party app store was established, users can install applications which would be registered in the app registry. The notification registry is a whitelist of Android package identifiers which the system will display phone notification from. This gives the user control over which notifications they would like to see from their phone. Miscellaneous user data can be saved into core data which is a local PostgreSQL database that stores key-value pairs. Each application's data is stored into a separate table which only that application can query through a handler-like interface.

2.3.5.7 Android Application

An Android application was developed to relay notifications for display on this device. The Android package consists of a background service which listens for system-wide notifications (with explicit

permission from the user) and an application frontend which mostly acts as a connection status indicator for debugging purposes. The listener service establishes a GATT server on startup and advertises itself as a BLE device named “notifier” which GlassOS will automatically connect to. It also initializes an instance of NotificationListenerService introduced in Android API 18 to receive the contents of notifications. When a notification arrives, the content is serialized into JSON format and sent to the GlassOS device where it will be passed upwards by the driver and handler. This functionality works passively and has low power consumption on both phone and device as it uses BLE standard.

2.4 Commissioning

2.4.1 Prototyping Testbed

Early on in development, as parts orders were arriving, it was necessary to immediately test a new component to ensure that it was not dead-on-arrival. In addition, the functionality, performance, and ability for the component to integrate into the system also needed to be verified so that alternatives can be pursued if this component turn out to be unsuitable for some reason. For these reasons, a barebone testbed as shown in Figure 20 was established. The figure includes all final components with the exception of the audio amplifier and bone conduction transducers. Once those parts arrived, it was simple to swap the original speakers for the new components and test their integration with the system and develop drivers. At any point when a component was malfunctioning, the integrity of the component could be verified by connecting it back onto this testbed.

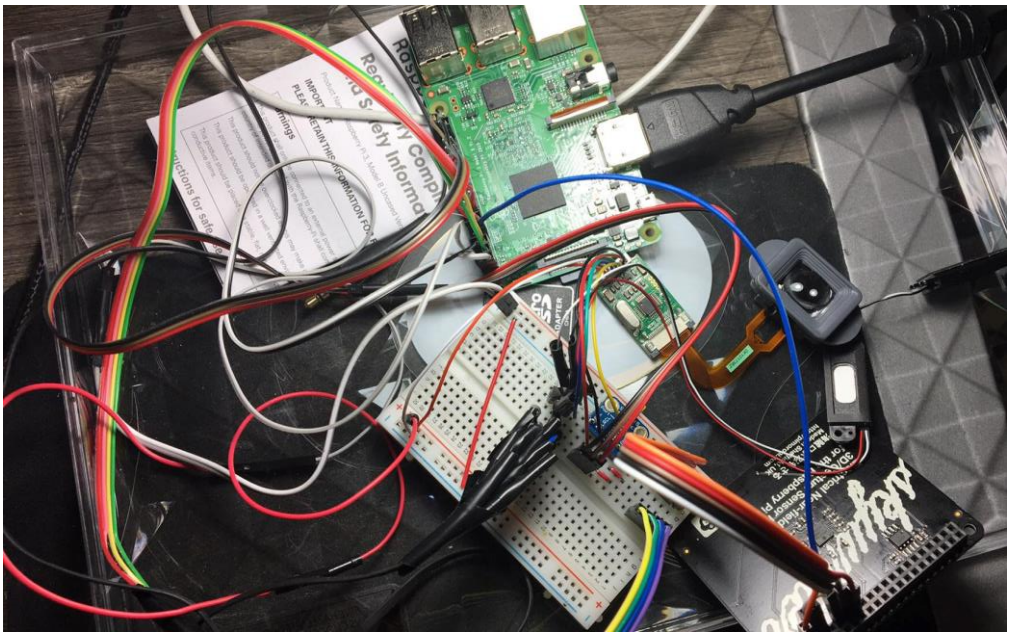


Figure 20: Prototyping testbed constructed on a breadboard including most final components.

2.4.2 Continuity Test

Once each component was verified to be working, they are fitted into their respective housing in device and connected to the central computing platform via custom made wires. Each wire is first checked for continuity between their terminals using a multimeter. Special care is taken for checking the 3.5mm phone connector as the soldering point for the 4 connector segments are very close in proximity and are highly susceptible to shorts. Each segment is checked again each other to ensure no shorts exist in the circuit.

2.4.3 Driver Testing

Once component connectively is verified, any further issues or malfunctions can be ruled as a software issue. Since component data passes through multiple software layers (driver, handler, and controller) which are individually susceptible to failure or bugs, code in each layer is manually tested separately in a unit test

manner. Table 2 shows the driver testing procedures for each component, which should be conducted in order as shown.

Table 2: Testing procedures to ensure correct function of component drivers.

Component	Testing Procedure
Display	1. Detach HDMI cable if attached 2. Ensure display powers on and displays full desktop image (not cutoff) 3. If cutoff exists, adjust overscan settings in `/boot/config.txt`
Gesture Sensor	1. Start the driver: `python ./src/drivers/touch.py` 2. Make several test gestures and verify console output matches gestures
Speakers	1. Run command: `speaker-test -c2 -t wav` 2. Verify speaker output on both channels
Microphone	1. Check the microphone device is connected and recognized: `arecord –l` 2. Create a test recording: `arecord --format=S16_LE --duration=5 --rate=16000 --file-type=raw out.raw` 3. Play the test recording to ensure microphone detected sound: `aplay --format=S16_LE --rate=16000 out.raw`
Assistant	1. Start the driver: `python ./src/drivers/assistant.py` 2. Test a command such as “Ok Google, what’s the weather?” 3. Verify console output has correct query and speakers play vocal response
Bluetooth	1. Start the driver: `python ./src/drivers/bluetooth.py` 2. Ensure the Android device is able to connect and send sample notification.

The test procedures above are executed in a terminal and thus are subject to slightly different runtime environment versus being spawned as a child process. Specifically, the standard output stream buffering behavior is different as explained in Section 2.3.5.3. To fully ensure compatibility with the associated handler and the rest of the GlassOS architecture, the driver handler is set to start with the main process to observe the behavior of the driver when spawned automatically (not by a user). Each handler contains logging code which reports child process events such as successful spawning, unexpected termination, and errors to the main process output stream. A successful startup should show a “service started” message for each component and no premature “service exited” messages.

2.4.4 Integration Testing

The final step in commissioning the device is to conduct integration tests of using combinations of components in the system to achieve software features. At this point lower level driver code and hardware issues should be ruled out. Any bugs can be triaged to the controller and view layer in the software. Table 3 shows several tests used and their procedures.

Table 3: Integration testing procedures.

Test	Components	Procedure
Gestures	Gesture Sensor	1. Swipe left and right at the home screen to browse through applications 2. Tap to open an application 3. Swipe up to close application 4. Swipe down to open notifications list
Assistant	Microphone Speaker Onboard WiFi	1. Trigger assistant and UI with “Ok Google” 2. Assistant UI should show voice query as text 3. Assistant should respond vocally 4. Assistant should open desired application when given an “open” command
Bluetooth	Bluetooth	1. Android device with companion app should connect automatically 2. Bluetooth icon should show in status bar when connected 3. Sending a sample notification from the companion app should be displayed

2.5 Performance Evaluation

2.5.1 Battery Life

Component power draw and theoretical runtime analysis conducted in Section 2.3.3.4 revealed that the integrated 2000 mAh LiPo battery should be capable of driving the device for up to four hours. Two experiments were conducted to obtain some real-world metrics on the battery life of the system. One experiment aims to determine the maximum runtime by running GlassOS idle at the home screen. Another experiment attempts to emulate typical usage by running the music app and continuously stream music which is played on the bone conduction transducers. The runtimes were recorded by a cron job which

executes every minute and updates a log file with the current time until the battery is exhausted and the device shuts down. The device was charged between trials until the charger LED lit green indicating that the battery is at or near maximum capacity. Table 4 shows the runtime results of these experiments.

Table 4: Battery life in different loading conditions.

Loading Condition	**Trial 1**	**Trial 2**	**Trial 3**	**Average**
Idle at Home Screen	209 minutes	201 minutes	206 minutes	205.3 minutes
Continuous Music Streaming	191 minutes	194 minutes	183 minutes	189.3 minutes

Although the device was unable to achieve the theoretical four hours of runtime, it was consistently able to pass three hours. Continuous activity such as music streaming does not appear to have a large impact on battery life. This may be due to an elevated base level of power consumption from running the main process and Electron, which essentially run a stripped-down version of Chromium. That is, even when idle, the power consumption is far above the Raspberry Pi's reported idle consumption of 260 mA. The approximately 16-minute loss in runtime could be due to battery consumption by the audio amplifier.

2.5.2 Bluetooth Stability and Range

One persistent issue with Bluetooth connectivity during development was the stability of the connection. Various tweaks and fallbacks were implemented to the Bluetooth driver in an effort to improve reliability. To test the final version of the driver, two test different test phones (OnePlus X and Sony Xperia XA1) were connected to the device and set to send a sample notification every hour overnight. Logging revealed the OnePlus X having disconnected 3 times over the course of 9 hours while the Sony Xperia XA1 disconnected over 300 times. Nonetheless, both devices automatically reconnected after each disconnection and was still connected when checked in the morning.

In addition, a separate test was conducted to determine the maximum range of Bluetooth connections. Shorter than specification range was expected given that the Bluetooth antenna sits within an ABS plastic housing and in proximity of live wires. The OnePlus X was able to maintain a connection up to about 10 meters from the device (with no obstacles in between). The Sony Xperia XA1, on the other hand, could hardly maintain a consistent connection at distances less than 1 meter from the device. The poor range result of the Sony Xperia XA1 correlates with the frequent disconnection observed in the other test. It is speculated that the unreliable connectivity is due to the Sony Xperia XA1 being a low-end device with an inferior Bluetooth chip. Overall, the device Bluetooth connectivity can only be concluded as adequate, but results vary with the connecting smartphone.

2.5.3 Overall Design Performance

The overall validity and performance of the final design can be evaluated by checking its fulfillment of the project constraints and criteria defined in Section 1.4 and 1.5. Five constraints were defined for limiting view obstruction, required hand interactions, physical size and weight, energy source, and component costs.

The final design uses an opaque display in front of one eye which does obstruct vision. The display area does not cover the entire field of view of one eye, so the constraint of less than 50% view obstruction is met. In addition, due to the overlapping field of vision between two eyes, the display appears translucent to the display eye. Overall, the view obstruction of this design is at optimal as can be without investing extra cost and development time and effort into achieving a polarizing beam splitter (PBS) display similar to the one on the Google Glass [13].

In terms of user hand interactions, the final design technically allows controlling the system purely through voice commands and no hand interactions at all. Even if the user chooses to manually manipulate the

interface, they can rely on simple, high error tolerance hand gestures which require significantly less accuracy and dexterity compared to a touchscreen. This constraint can be considered to be met.

The physical dimensions of the final design measures 220 mm by 260 mm by 97 mm when worn on the head with the elastic Velcro headband fully extended. The headband accommodates heads sizes up to 100 mm in radius (approximately 628 mm in circumference) when fully stretched, which is suitable considering the average and maximum Canadian male head size is 567 mm and 633 mm, respectively [14]. The device weighs 486.80 grams. Both the size and weight are within the constraint. It should be noted that this design deliberately positioned the Core Module on the back of the head to create an all-in-one device, but it could easily be detached and carried in a backpack to reduce the footprint and weight of the device on the user's head. This constraint can be considered met with possibilities for improvement.

The final design incorporates a moderately sized LiPo battery in the core module which allows the device to be powered for more than three hours in real world use cases. Thus, the portable energy source constraint is fulfilled.

Finally, the total component cost of the device comes to $266.31 CAD as shown in the Bill of Materials in Appendix E. The actual cost to produce this device may be slightly higher since the 3D printed components of the prototype were manufactured free of cost. Nonetheless, the total price would not have exceeded $300 CAD. This is far shy of the $1000 CAD cost limit imposed by the final constraint.

All in all, this final design was able to meet, if not exceed, all constraints. Given that the device cost is far below the cost constraint, there are improvements that can be made by simply upgrading some components. The area of improvement that would have the largest impact on the quality of the design would be to invest in either constructing or purchasing a PBS display, since the display is central to the user's experience with

the device. It is also possible to decrease the overall size and weight of the device by using a more compact compute platform solution or detaching the Core Module and placing it elsewhere. Lastly, the user experience is still dependent on hand interactions, but the current design contains the necessary hardware (e.g. microphone for smart assistant) to reduce dependency. Deeper and wider integration of voice control in the system can improve on this constraint given more time.

3.0 Schedule and Budget

3.1 Schedule

The span of this project lasted over two terms: 4A fall term and 4B winter term. In the MTE481 final report, which was written at the end of 4A, the schedule for 4A has already been discussed in depth. This section will instead take the projected schedule for 4B from the previous report, and discuss the actual schedule leading up the symposium on March 23.

3.1.1 Projected Schedule

Starting from the new year, there are 12 weeks up to and including the symposium week. We expected the display optics construction and the mobile companion app to be finished by the day of lecture start. The component interfacing drivers were expected to take longer, due to the sheer number of components we had to use. On the third week, the mobile developer could switch to the core software features and keep at it for four weeks. We also planned a final parts order on the fifth week, in case there were more parts needed. The next following weeks would be spent simultaneously miniaturizing the mechanical build and working on additional software features. Once the mechanical build had been finalized, the chassis would be manufactured through 3D printing. Once the software feature code has been built, the remaining time would be spent working out the bugs, up until the day of symposium. Finally, there is miscellaneous work related to the symposium that was planned for, such as the poster design and print, and demo video and other related prep work. The projected full schedule chart is shown below in Table 5.

Table 5: Projected Schedule for the weeks of January 1st to March 19th.

Task	1/1	1/8	1/15	1/22	1/29	2/5	2/12	2/19	2/26	3/5	3/12	3/19
Display Optics Construction												
Mobile Companion App												
Component Interfacing												
Core Software Features												
Final Parts Order												
Miniaturization & Optimization												
Additional Software Features												
Housing 3D Printing												
Software Bug Fixes												
Symposium Presentation												

3.1.2 Actual Schedule

Moving on to the real timeline, there were major changes that had to be done to the mechanical build due to the change from Raspberry Pi Zero to Pi 3. This change caused us to move up the planned optimization section and use it to design a new mechanical chassis. Some additional parts had to be ordered for this new build, so the final parts order was moved back to the 8th week so that we could confirm for all parts of the new build to be finalized. The actual schedule is shown below on Table 6.

Table 6: Actual schedule, deviations from projected in blue.

Task	1/1	1/8	1/15	1/22	1/29	2/5	2/12	2/19	2/26	3/5	3/12	3/19
Display Optics Construction												
Mobile Companion App												
Component Interfacing												
Core Software Features												
Final Parts Order												
Miniaturization & Optimization												
Additional Software Features												
Housing 3D Printing												
Software Bug Fixes												
Symposium Presentation												

3.2 Budget

When looking into the budget associated with this project, there are two main sections to discuss: the cost of the final product build and all expenditures used for this capstone project.

3.2.1 Final Assembly Cost

The financial constraint stated for the final design was to ensure that the bill of materials remained under $1000. This project exceeded its goals with a final build price of $266.31. The full BOM price table can be found in Appendix E.

The major cost saving decision was to use the Kitchener Public Library (KPL) 3D printing service. The service itself is free and the quality is up to par with WATiMake 3D printers (KPL uses Cubicon Single+ printers). This decision enabled us to save on printing costs, while avoiding the crowded schedule of WATiMake bookings near symposium date.

Every other electrical part is an off the shelf component. This avoided the cost of making custom PCB prints, since a one off custom board could cost more than what is found on the market.

3.2.2 Project Operational Expenditure

The amount to cover all expenses occurred from this project was projected to be $600. There are three group members in this team and, as determined by the MME department, each member is eligible for a $75 per term allowance. A subtotal of $450 would be provided by the university and the remaining $150 was determined from a three-way split of $50. This amount ($50) was the out of pocket value that each group member was willing to cover.

The resulting expenses occurred from the project turned out to be $1008.37. An expenditure report detailing an itemized list of purchases and costs has be included in Appendix F. When the total budget was conceived, it did not account for multiple iterations and variations in designs. Each different design during the research and development step required prototyping and testing a proof of concept in order to get consensus between group members. Each proof of concept prototype required different parts and, in summary, the costs added up.

The display portion of the project encompassed the majority of the parts related expenditures. In total there were three display configurations tested. The three setups and their costs were: a projector display ($151.85), a miniature TFT display with Fresnel lens magnification ($118.03), and a QVGA microdisplay

unit ($115.77). The display of choice is the QVGA microdisplay, which meant the other options were discarded aside for the sake of research and development. In retrospect, this is an expensive way to test proof of concepts for a project with a very limited budget. However, in our case, it would've been extremely difficult to confirm if these products would work as intended prior to their purchases, given their low availability of documentation.

Some parts were not available in Canada and had to be obtained from global suppliers. These global suppliers typically operate in USD, which is unfortunate since our operating budget's currency was set in CAD. During the eight months of this project, the CAD was experiencing a historic low point. The exchange rate of USD to CAD was approximately a 1.3 multiplier. Additionally, some part shipments were duty taxed on arrival. These parts, namely the QVGA microdisplay and the Fresnel lens, were the source of unexpected cost hikes.

Another reason for overshooting the budget was because the team had made a consensual decision to purchase duplicates of key components. For example, there were three Raspberry Pi 3 units in the projects: one carried over from a team member's inventory and two purchased new. This enabled improvements from two vectors. First, the redundancy meant that should a critical part fail, a replacement would be available without placing a new order and waiting on shipment. Secondly, it allowed all team members to develop in parallel by mirroring the environment image across all Pi(s).

Lastly, the MME department required the symposium poster to be sized at least 48"x36". A large format print of this size costed between $70 to $130 depending on the print material quality. In terms of the projected budget, it would cost 10% to 20% of the allocation. This was a non-trivial amount spent on an item not directly relevant to the product, but nonetheless necessary to the project.

4.0 Conclusion

This project sought to design an alternative to modern smartphones as mobile information terminals. Being able to free our attention as well as our hands from having to interact and hold smartphones leads to not only improved productivity and safety and a variety of real world tasks. Through an exploration of existing solutions to this problem, including a number of form factors and paradigms in the AR and VR industries, it was found that the best design direction to achieve the project goal was to create a head mounted display device.

An original design was presented in the MTE 481 Final Design Report. The design had a similar form factor to a pair of everyday eye glass. The small size of the original design was largely due to the use of the tiny Raspberry Pi Zero as the compute platform. Further testing revealed that the Pi Zero could not drive an ideal software stack and user experience. For this reason, the mechanical design had to be overhauled and the form factor increased to accommodate a larger and more powerful computing platform. A higher resolution yet smaller display was found which replaced the display proposed in the original design. An off-the-shelf gesture sensor replaced the proposed custom touchpad in the original design.

The revised mechanical design is based on a modular philosophy to speed up development time and reuse parts. There is a main frame that have multiple mounting screws for component mounting and it sits on the user's forehead. The left, right (encloses touch pad and MEMS microphone), and display enclosures are attached to the main frame. There are also slits for a Velcro elastic strap (with bone conduction speaker mounted) on the left and right enclosure that connect the back and front assembly. The back enclosure includes battery, Raspberry Pi 3, Audio Amplifier, and charging circuit. All enclosures were printed using ABS plastic and make use of M2.5 screws. The major components were also tested using finite element analysis. The mechanical components were fabricated through 3D printing using ABS plastic.

The device runs a custom operating system dubbed GlassOS on top of Raspbian Linux. The software architecture uses object-oriented programming concepts like encapsulation and abstraction to modularize the operating system into components that are individually maintainable. Hardware drivers of any language can be used with the system through a handler layer which translate the device specific data formats into a consistent internal representation. Users interact with the system through a high readability user interface controlled by simple swipe and tap gestures. Feature rich applications can be easily developed on top of this operating system through a consolidated API that allows utilization of hardware.

The prototype was brought to functionality through a commissioning process after all components were acquired. Firstly, each component was tested for functionality through a prototyping testbed. Each component's connection in the final prototype was checked for continuity. Once possible hardware issues were eliminated, two phases of software testing were conducted. The device drivers are first checked before a full integration test is ran on the live system.

The completed prototype fulfilled all constraints. It was able to achieve a host of features including simple gesture control and smart assistant integration that made it simple and non-distracting to use. Battery life testing revealed that the prototype was able to operate for over three hours on a single charge. The total cost of constructing the device was far below the cost constraint at only $266.31 CAD.

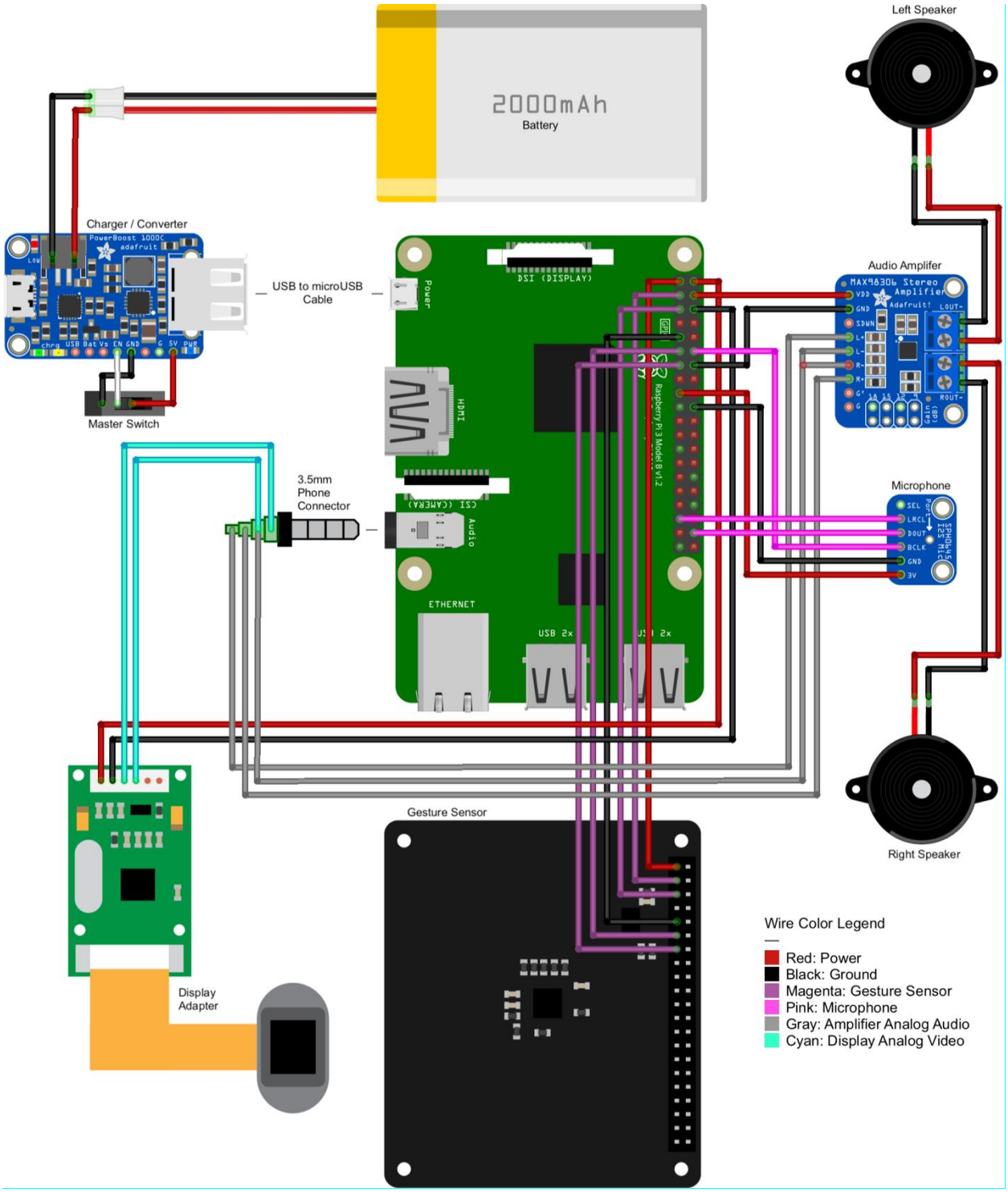
5.0 Recommendations

The final design was a large shift mechanically from the original design. The form factor increased significantly to accommodate the more powerful computing platform. For a wearable device, it is desirable to minimize footprint of the device so that it can better integrate into everyday life. Thus, a logical next step in iterating the design would be to miniaturize it. This can be done through a combination of using smaller alternatives for components and specializing the design. Currently, the form factor is dictated by the Raspberry Pi 3 Model B, which is also the largest component in the system. Smaller alternatives exist but may require more work and introduce more uncertainty in terms of developing software and device drivers. Another option would be to combine components onto a custom PCB which reduces size and wiring clutter. In addition, moving away from modular design and reducing size of each enclosure and the main frame now that the design is mature can also reduce the overall footprint of the device.

Another area of improvement for this design is the display. The current solution is a good baseline in terms of size, resolution, and quality. However, it is not translucent which causes some view obstruction to the user. To achieve a translucent display, one could use the reflector headset paradigm or invest in developing a PBS display similar to the one in the Google Glass. The reflector headset design involves using a visor style piece of glass to reflect a smartphone-sized screen placed overhead, which is cheap but ultimately not as compact as a PBS display.

A recommendation for the MME department would be to please have a cheaper large format printing service or subsidize an existing service. Every year there are hundreds of posters created for the sake of the engineering symposium. The cost of a large format print is non-trivial but yet predictable, and somehow the University does not provide any options to alleviate its financial burden.

Appendix A – Wiring Diagram



Appendix B – Final Design BOM and Cost

BOM Level	0	1	2	Supplier	Part Number	Part Name	Procurement Type	Qty	Cost/Per	Subtotal	Total Cost
0	-			In-House	001-A	Smart Glass	MTS	1		0	266.31
1		-		In-House	002-A	Smart Glass, Rail Assembly	MTS	1		0	
2			-	In-House	003-A	Right Housing w/ 1" Slit	MTS	1		0	
2			-	In-House	004-A	Right Housing Plexiglass Cover	MTS	1		0	
2			-	Digikey	1778-1082-ND	SKYWRITER HAT	OTS	1	$23.22	$23.22	
2			-	Digikey	1528-1908-ND	SPH0645LM4H I2S MICROPHONE	OTS	1	$9.54	$9.54	
2			-	In-House	005-A	Center Display Housing	MTS	1		0	
2			-	Banggood	912901	FPV QVGA 320X240	OTS	1	$77.24	$77.24	
2			-	In-House	006-A	Left Latch w/ 1" Slit	MTS	1		0	
1		-		In-House	007-A	Smart Glass, Core Assembly	MTS	1		0	
2			-	In-House	008-A	Core Unit Housing	MTS	1		0	
2			-	In-House	009-A	Core Unit Plexiglass Cover	MTS	1		0	
2			-	Amazon	B01CD5VC92	Raspberry Pi 3 Model B	OTS	1	$48.85	$48.85	
2			-	Sayal	1817-BA2A	LIPO 3.7V 2000MAH	OTS	1	$22.95	$22.95	
2			-	Digikey	1528-1349-ND	BOARD PWRBOOST1000 5V LIPO USB	OTS	1	$27.40	$27.40	
2			-	Digikey	1528-1381-ND	AMP AUDIO 2.8W CLASS D	OTS	1	$13.80	$13.80	
2			-	Digikey	1528-1948-ND	SPEAKER 8OHM 1W 90.1DB	OTS	2	$13.22	$26.44	
2			-	Sayal	GAS-1349	AUDIO PL 3.5 STEREO 4C RA B	OTS	1	$6.50	$6.50	
2			-	In-House	010-A	USB to Micro USB 5cm Long	MTS	1		0	
2			-	EMS	M2.5x25	Socket Head Capscrew M2.5x25	OTS	4	$0.26	$1.04	

1		-		In-House	011-A	Smart Glass, 1" Wide Elastic Band w/ Velcro	MTS	1	$5.64	$5.64	
1		-		Sayal	WAA-1565A	WIRE 24 AWG 20cm Long, BLK	MTS	8	$0.05	$0.40	
1		-		Sayal	WAA-1565A	WIRE 24 AWG 40cm Long, BLK	MTS	11	$0.10	$1.10	
1		-		Sayal	WAA-1565A	WIRE 24 AWG 60cm Long, BLK	MTS	5	$0.15	$0.75	
1		-		EMS	M2.5x20	Socket Head Capscrew M2.5x20	OTS	12	$0.12	$1.44	

Appendix C – Total Project Expenditure

Purchase Group	Supplier	Item	Quantity	Unit Price	Amount	**Group Subtotal**
1	Digikey	OCTOCAM PI ZERO W PROJECT KIT	1	$58.95	$58.95	**$341.62**
		0.96" 160X80 COLOR TFT DISPLAY W	1	$29.47	$29.47	
		SPEAKER 8OHM 1W 90.1DB	2	$13.22	$26.44	
		AMP AUDIO 2.8W CLASS D	1	$13.80	$13.80	
		MPR121 TOUCH SENSE 12-KEY	1	$11.03	$11.03	
		MIC MAG ANLG OMNI -39DB 9.7X4.5	1	$1.14	$1.14	
		SPH0645LM4H I2S MICROPHONE BOARD	1	$9.64	$9.64	
		DLP LIGHTCRAFTER DISPLAY 2000 EV	1	$151.85	$151.85	
		HST TAX	1	$39.30	$39.30	
2	Digikey	MAX98357A I2S AMP BREAKOUT BOARD	1	$8.65	$8.65	**$44.01**
		SKYWRITER HAT (INDIVIDUAL)	1	$23.22	$23.22	
		SHIPPING	1	$8.00	$8.00	
		HST TAX	1	$4.14	$4.14	
3	Digikey	BOARD PWRBOOST1000 5V LIPO USB	1	$27.40	$27.40	**$49.74**
		SPH0645LM4H I2S MICROPHONE BOARD	1	$9.54	$9.54	
		SHIPPING	1	$8.00	$8.00	
		HST TAX	1	$4.80	$4.80	
4	Sayal Electronics	TUBING WRAP	1	$1.76	$1.76	**$54.69**
		WIRE 24 AWG	1	$7.29	$7.29	
		LIPO 3.7V 2000MAH	1	$22.95	$22.95	
		CONN PLUG CRIMP M	1	$4.95	$4.95	
		CONN PLUG CRIMP F	1	$4.95	$4.95	
		AUDIO PLUG STEREO 4C	1	$6.50	$6.50	
		HST TAX	1	$6.29	$6.29	

5	Banggood	FPV QVGA 320X240	1	$77.24	$77.24	**$115.77**
		SHIPPING	1	$17.37	$17.37	
		DUTY	1	$21.16	$21.16	
6	Edmund Optics	FRESNEL ASPH 1.0 X 1.0IN	1	$34.00	$34.00	**$88.56**
		SHIPPING	1	$30.78	$30.78	
		DUTY	1	$23.78	$23.78	
7	BuyaPi	RASPBERRY PI ZERO W	1	$13.45	$13.45	**$27.63**
		SHIPPING	1	$11.00	$11.00	
		HST TAX	1	$3.18	$3.18	
8	Amazon	Raspberry Pi 3 Model B	1	$48.85	$48.85	**$68.70**
		CanaKit 2.5A Raspberry Pi Micro USB Power Supply	1	$11.95	$11.95	
		HST TAX	1	$7.90	$7.90	
9	Canada Computers	USB Adapter	1	$4.99	$4.99	**$5.64**
		HST TAX	1	$0.65	$0.65	
10	Fabricland	VELCRO	1	$1.84	$1.84	**$6.37**
		ELASTIC BAND	1	$3.80	$3.80	
		HST TAX	1	$0.73	$0.73	
11	Live Switchboard	Poster	1	$94.92	$94.92	**$94.92**
12	EMS	Socket Head Capscrew M2.5x5	20	$0.58	$11.60	**$31.19**
		Socket Head Capscrew M2.5x10	20	$0.38	$7.60	
		Socket Head Capscrew M2.5x20	20	$0.12	$2.40	
		Socket Head Capscrew M2.5x25	20	$0.26	$5.20	
		Hex Nut M2.5	20	$0.04	$0.80	
		HST TAX	1	$3.59	$3.59	
13	Amazon	Raspberry Pi 3 Model B	1	$48.85	$48.85	**48.85**
14	Staples	Kingston 16GB microSD Card	1	$15.82	$15.82	**15.82**
15	Staples	Lexar 16GB microSD Card	1	$14.86	$14.86	**14.86**
					Total Amount	**$1,008.37**

Works Cited

[1] Microchip Technology Inc., "MGC3030/3130 3D Tracking and Gesture Controller Data Sheet," [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/40001667E.pdf. [Accessed 27 March 2018].

[2] Raspberry Pi Foundation, "Raspberry Pi 3 Model B," Raspberry Pi Foundation, [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed 30 March 2018].

[3] B. F. Spenner, "FLCOS hi-res microdisplays, faster than LCD, for video and holographic memory applications," AspenCore LLC, 22 December 2006. [Online]. Available: https://www.embedded.com/design/prototyping-and-development/4013117/FLCOS-hi-res-microdisplays-faster-than-LCD-for-video-and-holographic-memory-applications. [Accessed 29 March 2018].

[4] J. Strickland, "How Vibration Speakers Work," System1, [Online]. Available: https://electronics.howstuffworks.com/gadgets/audio-music/vibration-speakers.htm. [Accessed 29 March 2018].

[5] Adafruit Industries, "Bone Conductor Transducer with Wires - 8 Ohm 1 Watt," Adafruit Industries, [Online]. Available: https://www.adafruit.com/product/1674. [Accessed 29 March 2018].

[6] Adafruit Industries, "Stereo 3.7W Class D Audio Amplifier - MAX98306," Adafruit Industries, [Online]. Available: https://www.adafruit.com/product/987. [Accessed 29 March 2018].

[7] R. Tran, "How to Select the Right Battery for Wearables & Consumer Electronics," Advantage Business Media, 2 August 2016. [Online]. Available: https://www.pddnet.com/article/2016/08/how-select-right-battery-wearables-consumer-electronics. [Accessed 29 March 2018].

[8] J. Geerling, "Power Consumption," [Online]. Available: https://www.pidramble.com/wiki/benchmarks/power-consumption. [Accessed 30 March 2018].

[9] Banggood, "FPV Night Vision QVGA 320X240 Monocular Goggles Viewfinder Monitor Micro Display For FPV RC Drone," Banggood, [Online]. Available: https://www.banggood.com/FPV-Night-Vision-QVGA-320X240-Monocular-Goggles-Viewfinder-Monitor-Micro-Display-p-912901.html?rmmds=detail-left-hotproducts__2&cur_warehouse=CN. [Accessed 30 March 2018].

[10] Knowles Electronics, "SPW2430HR5H-B," 2014. [Online]. Available: http://www.knowles.com/eng/content/download/5642/89383/version/8/file/SPW2430HR5H-B.pdf. [Accessed 30 March 2018].

[11] Adafruit Industries, "Adafruit Powerboost 1000C," Adafruit Industries, [Online]. Available: https://learn.adafruit.com/adafruit-powerboost-1000c-load-share-usb-charge-boost. [Accessed 30 March 2018].

[12] E. Klitzke, "Stdout Buffering," 23 December 2016. [Online]. Available: https://eklitzke.org/stdout-buffering. [Accessed 30 March 2018].

[13] S. Hazra, "How does Google Glass project the image onto the glass?," Quora, 21 September 2015. [Online]. Available: https://www.quora.com/How-does-Google-Glass-project-the-image-onto-the-glass. [Accessed 2 April 2018].

[14] A. D. Nguyen, A. A. Simard-Meilleur, C. Berthiaume, R. Godbout and L. Mottron, "Head Circumference in Canadian Male Adults: Development of a Normalized Chart," 2012. [Online]. Available: https://pdfs.semanticscholar.org/a28d/35f6e2af7347861a808eeb1346f13ac9bf9c.pdf. [Accessed 3 April 2018].