Volumetric Display Using a Stereoscopic Head Tracking System

EEL4914 Group 14

Dane Bouchie, Cp. E.
Matthew Hosken, Cp. E.
Colburn Schacht, E. E.
Eric Smithson, Cp. E.
# Table of Contents

1. Executive Summary 6

2. Product Description 7
   2.1 Motivation 7
   2.2 Objective 8
   2.3 Product Criteria (High-Level Requirements) 9
   2.4 Technical Specifications 9
   2.5 House of Quality 10
   2.6 High Level Diagram 11

3. Research, Technologies, and Components 14
   3.1 Similar Products 14
      3.1.1 VR systems 14
      3.1.2 AR systems 15
      3.1.3 “Tabletop” Holographic Display Systems 15
   3.2 Serial Communication Protocols 16
      3.2.1 UART 16
      3.2.2 I2C 17
      3.2.3 SPI 18
   3.3 Processing Units 18
   3.4 Considered Display Methods 19
      3.4.2 Glasses-Free 3D 20
      3.4.3 Pepper’s Ghost Illusion 20
      3.4.4 Decision 20
   3.5 Hardware Requirements 21
      3.5.1 Projector 21
         3.5.1.1 DLP Projectors 21
         3.5.1.2 3LCD Projectors 22
         3.5.1.3 LCoS Projectors 23
         3.5.1.4 Projector Decision 23
      3.5.2 Infrared LEDs 24
      3.5.3 Transmitter/Receiver between Display and Glasses 26
         3.5.3.1 IR Communication 26
         3.5.3.2 DLP-Link 28
         3.5.3.3 RF Communication 29
         3.5.2.4 Communication Decision 30
      3.5.4 Camera System/Data Transmission 30


3.9 Application Design System 47
  3.9.1 Viability of other Gaming Engines 49
  3.9.2 Camera Techniques in 3D simulations 50
  3.9.3 Control Scheme 50
  3.9.4 Necessary Computer Specifications 51

4. Standards and Design Constraints 53
  4.1 Standards 53
    4.1.1 Indian Hill C Style and Coding Standards 53
  4.2 Health and Safety Concerns 54
    4.2.1 Motion Sickness 54
    4.2.2 Eye Strain 56
    4.2.3 Photosensitive Epilepsy (PSE) 57

5. Design 58
  5.1 Overview 58
    5.1.1 Glasses Detailed Overview 59
    5.1.3 Rendering System Detailed Overview 61
  5.2 3-D Effect and Head Tracking Concept 62
    5.2.1 Projection Algorithm 63
    5.2.2 Head Tracking Algorithm 65
      5.2.2.1 Kalman Filter 68
    5.2.3 IR Point Tracking Algorithm 69
      5.2.3.1 Camera Details 73
    5.2.4 Data Serialization and Interoperability 73
      5.2.4.1 Start/Stop 74
      5.2.4.2 Error/Check 74
      5.2.4.3 Data Types and Structures 74
    5.2.5 Overall Calibration Process 75
  5.3 Hardware Design 76
    5.3.1 Final Schematic 77
    5.3.2 Printed Circuit Boards 79
    5.3.3 Hardware Parts List 81
  5.4 Software Requirements 81
    5.4.1 Universal Game Idea 81
    5.4.2 Nvidia 3D Vision drivers 82
    5.4.3 Feasibility of Implementation 84
    5.4.4 Parsing Data from the IR cameras 85
5.4.5 Limitations of the Software Vs Hardware
5.4.6 Ability to use Separate Operating System
5.5 Development Environment
5.5.1 Unity
5.5.2 Visual Studio Community 2017
5.5.3 C#
5.5.4 Source Control
  5.5.4.1 Git
  5.5.4.2 Github
  5.5.4.3 Source Control Workflow
5.6 Application Design
  5.6.1 Game structure
  5.6.2 Game components and design goals.
  5.6.3 Game Description
5.7 Limitations on the Design

6. Testing
  6.1 Software Testing
     6.1.1 Unit Testing
     6.1.2 Integration Testing
  6.2 Hardware Testing
     6.2.1 Component Hardware Testing
        6.2.1.1 Power Supply Circuit
        6.2.1.2 Battery
        6.2.1.3 DLP-Link Communication
        6.2.1.4 IR LED Communication
        6.2.1.5 Liquid Crystal Lense Operation
     6.2.2 Hardware Test Procedures
        6.2.2.1 Power Supply Circuit Test Procedure
        6.2.2.2 Battery Test Procedure
        6.2.2.4 IR LED Communication Test Procedure
        6.2.2.5 Liquid Crystal Lense Operation Test Procedure
  6.2.3 Breadboard Testing

7. Administrative Content
  7.1 Task Division
  7.2 Project Milestones
  7.3 Budget and Financing
8. Conclusion 122
  8.1 Improvements 122
  8.2 Marketability 123
  8.3 Concluding Thoughts 124

9. Implementation 125
  8.1 HeadTracking 125
  8.2 Interfacing 126
  8.3 Application Engine 126
  8.4 Hardware 128
  8.5 Improvements 129
  8.6 Final Thoughts 130

Appendix A - References 132

Appendix B - Copyright Permissions 133
  OpenCV (Including Documentation) 133

Appendix C - Software code 135
  C# DLL Code 135
1. Executive Summary

This project is tasked at designing a holographic system which renders 3D effects to create a virtual environment embedded inside of a display surface. This is accomplished by building several subsystems. The first is a stereoscopic display system using glasses, which uses our own implementation of DLP-Link to simultaneously display 3D images. The second is a head-tracking system using a combination of infrared LED points with IR cameras, and accelerometer and gyroscopic are processed using a sensor-fusion algorithm for accurate tracking of the user. A rendering system takes its known orientation and the processed orientation of the user to render stereoscopic, perspective-transformed images to create the full 3D virtual environment effect. An overall data transmission system including serial ports, and wireless infrared transmission creates data flow between each system. Finally, applications were designed to demonstrate the system's ability and features.

Interaction of the system involves a user wearing the aforementioned glasses to visually produce the virtual environment. They will then be able to interact with the virtual environment through a human interface device such as a joystick. Applications will explore interaction in three focused areas: entertainment, design, and multiple-user.

Although some approaches to the criteria already exist, the overall specific implementation of the stereoscopic head-tracking with this level of interaction is yet to be seen. As such, our objective is a proof-of concept design. Market value was excluded from thought to allow freedom to focus on quality of design, and research needs. Our objective is to exercise and prove electrical engineering, computer engineering, and computer science skills. Given our existing skillset, we were able to apply it towards a design and see it constructed. We have given ourselves criteria and constraints to meet throughout this project. Each of the systems were researched, evaluated and conceptualized until solutions were found that meet the criteria of the design.

The device will be constructed by research and selection of system software, software libraries, hardware systems and hardware components. We will have carefully designed software applications, system integration, sensor fusion algorithms, and hardware which includes a schematic, PCB design, and printed PCB assembly.
2. Product Description

The holographic system gives the user a unique experience to a virtual environment. It allows them to explore interaction will a full 3D system without intruding on the user’s common interaction with the real world. Applications will be explored in entertainment purposes which provide a unique experience, design application which can provide a more detailed visuals than traditional 2D environments can produce, and multi-user interaction for benefit to both.

In this section we will discuss the background and motivation behind the product. To define a strong design, we created criteria and specifications to meet that are further detailed. Finally, we will list the constraints of the system during the design.

2.1 Motivation

Virtual and Augmented Reality are two highly-anticipated technologies being researched and marketed today. Virtual Reality (VR) systems put the user inside a virtual environment to simulate new realities or provide easy accessible or simplified existing environments. Augmented Reality (AR) takes aim at adding features to our everyday existing reality such as notifications, virtual objects, and other ideas. Both methods use 3-dimensional illusions through head-tracking and stereoscopic imagery to provide real-time perspective which emulates object interaction in the real world.

Most implementations of VR/AR feature headsets such as HTC’s Vive and Microsoft’s HoloLens. In this project, we provide a different solution featuring a virtual display environment embedded in a display surface. Rather than display images overlaid over a user’s eyes, we explore applications in a 3-dimensional physical display object. And in addition, provide an experience where multiple users can interact with a single synchronized environment.

This device provides research in design and a platform for future research in the topic. Majority of motivation comes as an academic exercise and to create a unique experience. Due to relatively high expected costs, it is unlikely to be marketed. It is not targeted as a specific solution, but rather a proof-of-concept design.

It is noteworthy to acknowledgement the existence of similar “table-top” systems which use head-tracking technology on a display surface. To contrast our design, our system integrates further steps that provides higher quality and more responsive headtracking, stereoscopy for the complete 3D effect, and overall, a base which does not interfere with the display surface view.
Overall, we wanted to provide a system that allows users to have a VR/AR experience without the restraint of a full headset. Tech explored here could transition to head tracking applications in both commercial and entertainment applications.

2.2 Objective

In general our project isn't as much of a problem solver. However, we feel that what we are doing is a natural progression in the media industry. In essence we are taking years or even decades of research and prototyping into our design while combining and refining the ability to view a three dimensional object where there isn't one.

Our concepts are that it is possible to have a three dimensional viewed object just as immersive and real as the current generation of virtual reality provides, and to improve the technology and update out the bugs and imperfections. This project is an evolutionary step, and we are fully aware that in five years what we do might possibly become obsolete, but we must learn to walk before we can learn to run.

First, we will determine and prove that it is possible to track an object to within a human's subconscious vision's ability to determine an anomaly in its observation. In other words exceptionally fast and precise head tracking within a small area and with a component that could be fitted comfortably onto a pair of standard 3D glasses.

Second, we will demonstrate that true 3D objects, as opposed to the 2.5D Doom-like graphics projection as discussed earlier, is possible using standard stereoscopic 3D technology. It could be argued that this is very opinion-based and can only truly be understood by each individual user, however a grand majority of opinions expressed has come to the assumption that current stereoscopic 3D looks eerily similar to looking at a flat TV screen while someone holds a shopping list in front of your face with a fishing rod. It's flat, it's not immersive, and it’s sometimes kind of nauseating when the image doesn’t change when you move your point of view. We will prove that it is possible to have both the first concept and “angle of attack” based changes in viewing the image or object can coexist without the complete separation of the virtual world and the living world as the HTC Vive does.

All three of these concepts are easily implemented individually, but combining them is the challenging part. Currently the line between what is real and what is virtual is both thin, and blurry. The harder we push what is virtual toward the line of what is real, the more forcefully the environmental and technological limitations push back. I suppose instead of solving a problem, we are in effect
changing the answer to the all-important question: “How far can we go?” And instead of getting our project close to the limits, forcing the limits to be moved further than they were before.

2.3 Product Criteria (High-Level Requirements)

The following are criteria which provide a list of high-level functionality requirements of the product to design.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>The device will simulate the viewing of a 3-dimensional image within the confines of the display surface.</td>
</tr>
<tr>
<td>1.1</td>
<td>At least 2 users can interact with the same device simultaneously.</td>
</tr>
<tr>
<td>1.2</td>
<td>When used properly and for shorter durations, the device will not cause motion induced nausea for most users.</td>
</tr>
<tr>
<td>1.3</td>
<td>The device will be open ended enough to provide applications to innovative developers.</td>
</tr>
<tr>
<td>1.4</td>
<td>The product will run at least one program demonstrating a design application.</td>
</tr>
<tr>
<td>1.5</td>
<td>The product will run at least one program demonstrating an entertainment application.</td>
</tr>
<tr>
<td>1.6</td>
<td>The product will run at least one program which features at least two users interacting together simultaneously.</td>
</tr>
</tbody>
</table>

Table 1 High-Level Requirements

2.4 Technical Specifications

The following is a list of technical specifications we determined to meet the product criteria with a high-quality design.
2.0 The display shape will be no larger than 1 cubic meter.
2.1 The display will have a pixel density higher than 9 dpi.
2.2 The display will run at a minimum of 20 frames per second refresh rate.
2.3 The head tracking system will track position within a standard deviation of 10 cm.
2.4 The head tracking system will track within an area of 0.25 m².
2.5 The head tracking system will be responsive to 250 ms.
2.6 The device will cost no more than $3000.

<table>
<thead>
<tr>
<th>2.0</th>
<th>The display shape will be no larger than 1 cubic meter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The display will have a pixel density higher than 9 dpi.</td>
</tr>
<tr>
<td>2.2</td>
<td>The display will run at a minimum of 20 frames per second refresh rate.</td>
</tr>
<tr>
<td>2.3</td>
<td>The head tracking system will track position within a standard deviation of 10 cm.</td>
</tr>
<tr>
<td>2.4</td>
<td>The head tracking system will track within an area of 0.25 m².</td>
</tr>
<tr>
<td>2.5</td>
<td>The head tracking system will be responsive to 250 ms.</td>
</tr>
<tr>
<td>2.6</td>
<td>The device will cost no more than $3000.</td>
</tr>
</tbody>
</table>

Table 2: Specifications and Engineer Requirements

2.5 House of Quality

The criteria and specifications described in the previous sub-sections allow us to consider tradeoffs between various qualities in the technical specifications and constraints. This can be arranged into a house of quality to visualize and focus on the most important aspects that need to be considered in the design.

The following key is used when addressing the table:
- ↑: Correlation
- ↑↑: Strong correlation
- ↓: Inverse correlation
- ↓↓: Strong inverse correlation
- +: Positive effect on quality
- -: Negative effect on quality
Overall, we can see the projection quality is based most upon pixel density and the refresh rate. This is also inversely proportional to the cost which needs to budgeted. (This is discussed later in constraints). From here, the variety of platforms depends upon our methods used when developing the applications. The ease of installation depends upon our display size. Finally the high factor in the cost are the display characteristics themselves.

2.6 High Level Diagram

The following diagram provides a high-level overview of the system design. To begin, User Head movements physically move the attached infrared leds, gyroscope and accelerometer. The infrared camera then tracks the updated position of the infrared LEDs. Parallel to this, gyroscope and accelerometer data are transmitted via an IR transmitter to a receiver in the tracking system processor. The processor then performs sensor fusion algorithms to determine an accurate orientation of a user. Once calculated, this information is transmitted to the rendering device driver via a us serial port. The driver then passes this directly to the application when requested. Finally, the rendering
system renders the appropriate projection given the user’s orientation. This is then displayed as an active 3D image, and filtered appropriately to the user’s left and right eye’s, completing the 3D effect.

To setup the device, a calibration processes is needed. This exact process is later discussed in the Calibration subsection. For completeness, the application controls the initiation of the calibration process. When requested this request is feed into the driver, and sent to the processor over the serial port. The processor can than perform the calibration process and exchange further data with the rendering system.
Figure 2: Project Block Diagram
3. Research, Technologies, and Components

In the following section we will go over research efforts lead to the decision of the overall architecture of the design. We will then expand into selection and comparison of components that will make up the system. These include hardware components, hardware systems, software architectures, algorithms, and software libraries.

3.1 Similar Products

Given the uniqueness of our task, there are not existing systems which we can directly draw design from. The proof-of-concept initiative gives us the challenge of creating majority of the systems ourselves. Instead, we gain insight from applications in the Virtual Reality (VR) and Augmented Reality (AR) fields.

3.1.1 VR systems

The Oculus Rift and the HTC Vive, currently the leading products in virtual reality technology for interested middle class or higher consumers, both place large screens directly in front of the user's eyes, with foam all around the headset in order to block out the light from coming into the viewing aperture from the living world. We wish to prove that it is not necessary to divide the living world and the virtual world, but that it is possible for them to coexist.

The biggest difference between the current accepted generation of virtual reality technology and what we hope to create is the environment in which the image is viewed. The term hologram actually describes specifically an object or image that is projected and can be viewed in the living world, in which we all exist, and appear as if it was part of the living world itself.

Currently the smallest extreme precision head tracking is best demonstrated by the HTC Vive headset, which must be fitted to the user's skull via straps and seals the eyes and part of the nose inside a small chamber with the projecting screens. This is uncomfortable, and constrains the user's use time. Our design will prove that high quality head tracking does not require the user to be subjected to such an environment.

Since the display methods in headsets vary greatly from our application, the most applicable design concepts are those in the head-tracking systems. The HTC Vive uses a fairly complex head-tracking system. HTC named their technology Lighthouse systems. Each Lighthouse base needs to be placed in a particular configuration in the room. The base then contains a laser system that sweeps across the room. When the laser hits a photosensor on the headset or
on the controllers, time is measured in reference to the lighthouse’s camera start of a laser sweep. Using this information, the relative angle to the lighthouse can be precisely measured. Collectively, the position can be found using triangulation. Although this system has been proven to be very effective, the optical and mechanical knowledge required for the laser-sweep system is out of our expertise for our team alone.

A different approach is the Oculus Rift’s Constellation system. The Constellation system is composed of infrared LED’s which can be tracked using an infrared sensor. As the user changes their orientation, the LED’s position in reference to the sensor updates. Using transformations and the known positions of the tracking points, the final orientation can be calculated. It is from this inspiration in the head tracking system we draw upon.

### 3.1.2 AR systems

Accelerated Reality (AR) is aimed at a slightly different objective than VR. Whereas VR creates an entire virtual environment for the user, AR typically focuses on enhancing reality with virtual objects. Headset applications of AR such as Microsoft’s HoloLens, are created in a similar to VR, except overlaying the images on the user’s view as opposed to replacing it. Another implementation of AR is using a live camera system to overlay virtual object’s. This system provides only a flat 2D viewport, but still adds features to reality.

To perform head-tracking, or rather orientation Microsoft’s HoloLens uses a depth camera to perform spatial mapping on the real world. This in turn, is used to construct the correct viewpoints of the virtual objects on the display.

Unfortunately, existing AR systems do not contribute much for our application. AR systems are focused on image registration and visual odometry to determine how to render an object. Our system, requires higher precision than are used in typical AR applications, and spatial mapping is redundant in this manner.

Overall, our system technically falls into a category called spatial augmented reality. These systems use projection mapping to change the surface appearance, and produce visual effects on the display surface.

### 3.1.3 “Tabletop” Holographic Display Systems

Recently, a small market for “tabletop” holographic systems have appeared. The systems are designed similarly to how we have described our criteria. A display surface embeds a virtual environment into the surface. This technology makes use of an illusion called Pepper’s ghost by reflection a 2D display above the 3D
display shape to render the appearance of the 2D display inside the 3D display surface. (Further described in Section 4.4)

While this technology produces an interesting effect, it is important to note a drastic difference between our system and these devices. Without head-tracking, these devices do not produce perspective-changing imagery. They only provide the illusion that a virtual object is inside the display shape, and unfortunately do not provide a realistic interaction similar to the real world. The Pepper's Ghost illusion can only provide depth to what is reflect, which is only a 2D screen. In addition to the lack of head-tracking, they also lack the stereoscopy to produce stereoscopic depth for both eyes. Even more disheartening, it is clear they use clever video editing to fake the full effect in their advertising videos.

That being said, these systems still have important factors we can gain insight from. In particular, the with the way they interact with the users. Most devices feature a pyramidal shape. This is particularly useful in that it defines boundaries for each user interacting with the device. The display is naturally divided. This is an important factor for our device since only one perspective can be shown on a single portion of the device.

Overall, even though inspiration did not come from these devices, we wish to expand upon their intent. Implementing stereoscopy and head tracking creates the necessary capabilities for a full volumetric display.

### 3.2 Serial Communication Protocols

For all of the communication between our main control units and the ancillary devices we are using we will need to utilize one of the major communication protocols. In the following sections we will be going over UART, I²C, and SPI. Throughout each of them we will be determining their pros and cons and deciding which would fit our needs most completely.

#### 3.2.1 UART

A universal asynchronous receiver/transmitter (UART) is a computer hardware device for asynchronous serial communication in which data format and transmission speeds are configurable. A UART is usually part of an integrated circuit (IC) used for serial communications over a peripheral device serial port. The UART takes bytes of data and transmits the individual bits in a sequential fashion. A second receiver UART then re-assembles the bits into complete bytes.

The idle state of UART is high voltage. Each character of data (1 byte) is framed between one low start bit, and one high stop bit. Other non-data bits include a
parity bit, which would come after the data bits. The data bits are transmitted in order from lowest significance to highest significance.

All operations of UART are controlled by a clock signal which runs typically at 8 times the bit rate. Sophisticated UARTs synchronize on every transferred bit. Bits are read into a shift register as they come in. When an entire character is successfully received, the receiving UART may send an interrupt to the receiving system so the system will grab the new data before the next byte comes in. Some receivers have a FIFO buffer in memory to give the receiving system more time to drain the buffer at a convenient time, depending on the application.

Our application requires received data be received and processed as soon as it becomes available for the smoothest experience, so we will have to configure our hardware accordingly. Data loss due to receiver lag isn’t a huge concern.

UART transmitting is easier in that the transmitter can just “fire and forget” bits with no regard to error checking (except for the generation of the parity bit), or determining timing from the line state.

3.2.2 I²C

I²C (Inter-Integrated Circuit) is a multi-master, multi-slave packet switched, single-ended, serial computer bus. It is used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.

I²C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock Line. The I2C reference design has 7-bit or 10-bit address space. The reference design is a bus with a clock (SCL) and data(SDA) lines with 7-bit addressing. The bus has two roles for nodes: master and slave.

- **Master node** - node that generates clock signal and initiates communication with slaves.
- **Slave node** - node which received clock signal and responds when addressed by the master.

Multiple master nodes can be present. Master and slave roles can be changed between messages. There may be four potential modes of operation for any given bus device.

- **Master Transmit** - master node sending data to a slave
- **Master Receive** - master node receiving data from a slave
- **Slave Transmit** - slave node sending data to the master
- **Slave Receive** - slave node is receiving data from the master
I2C is addressed, which means any master in transit mode starts by sending a start bit followed by the 7-bit address of the slave it wants to communicate with, followed by a single directional bit which indicates whether it wants to read (1) or write (0) from/to the slave. If the slave exists then it will respond with an ACK bit (active low).

3.2.3 SPI

The Serial Peripheral Interface bus (SPI) is a synchronous serial communication interface specification used for short distance communication. SPI devices communicate in full duplex mode using a master-slave paradigm with a single master. SPI is sometimes called a four-wire serial bus.

Pros
- Full duplex communication is default for this protocol
- Push-pull drivers (as opposed to open drain) provide better signal integrity.
- Higher throughput than I2C.
- Not limited to 8-bit words
- Very simple hardware interfacing
- Lower power requirements than I2C
- Slaves use master’s clock and do not need precision oscillators
- Slaves do not need a unique address
- Uses less board real estate (important for us)

Cons
- Requires more pins on IC packages than I2C
- No hardware flow control by the slave
- No hardware slave acknowledgement
- Only supports one master device
- No error-checking protocol defined

3.3 Processing Units

The processing for our project will be split into 3 main sections, each run by a separate piece of hardware. The three sections will be the glasses, the transmitter/receiver portion, and the head tracking/game software portion. The three different pieces of hardware that will control these different sections will be an MSP430 for the glasses, a Raspberry Pi Zero for the transmitter/receiver, and unity for the application.

We chose to split our computations and processing between these three different units because everything necessary for this project to work would be too intensive and complex to implement into just one piece of hardware. By splitting our tasks between these different units we believe we can implement a
system that most effectively uses the resources we have and offer the best final product.

3.4 Considered Display Methods

Before we chose the display method that we are utilizing for our project, we considered a couple other methods of developing a 3D effect. The main two other methods that we did some research on before deciding on active shutter 3D were glasses-free 3D and the use of a pepper’s ghost illusion.

3.4.1 3D TV/Projector

3D film has been around for many decades. The 3D film effect can be attributed to the use of stereoscopic imaging which is when two photographs of the same object taken at slightly different angles are viewed together, creating an impression of depth and solidity. This process is presently achieved through two different methods: Passive 3D and Active 3D. Both of these types of 3D are available in many of the 3D TV and Projector units out today.

The first type of 3D imaging is passive 3D film. Film or images are made to look 3d by displaying a single frame through 2 separate shots. The images are overlaid and displayed in one picture with each corresponding line of pixels belonging to the opposite photo, and each of these photos are also shown at different wavelengths of light. The viewer then uses polarized glasses that filter out lines of pixels meant for the opposite eye, thus giving you two separate images for each individual eye.

The newer method for producing the stereoscopic effect in 3D film is active 3D or the active shutter system. Due to the advancement of LCD and television technology this method has become possible. Active 3D is a more complex method of reaching the desired effect, however, is known to produce a superior quality viewing experience. Rather than overlaying images to create a single frame, full images of each viewpoint are flashed onto the screen simultaneously at a high enough frequency to trick the eye into thinking a single image is displayed. As each image is being displayed, the viewer is wearing a pair of specially designed 3D glasses that can electronically black out the eye that does not correspond to the image on the screen. This “black out” effect is produced through the use of LCD lenses.

When we compare these two methods, one big thing stands out: complexity. The passive filter system is extremely simplistic in nature and only requires manipulation of the image being displayed. This simplicity also factors into the lower overall cost of using such a system. The active shutter system is a lot more complex, however not without reason. Active 3D is known for creating a better quality viewing experience at the minimal chance that you may
subconsciously notice the flickering and get a headache after extended amounts of time using them.

3.4.2 Glasses-Free 3D

Nowadays companies are beginning to showcase the new glasses-free 3D technology. The fundamental method of using stereoscopic images is still there, however, moved from the users eyes to the screen itself. This is achieved using a process called autostereoscopy. At the moment the main way to achieve autostereoscopic imaging is through the use of a parallax barrier or using integral photography and lenticular rays. Although this would be interesting and eliminate the need for glasses, the technology is still being researched and the availability of resources to create such a display are far out of reach for us.

3.4.3 Pepper’s Ghost Illusion

Pepper’s Ghost is an Illusion first popularized by the namesake playwright, where an image is projected into another 3D space via reflection. This image is a reflected illusion produced by a large piece of glass or reflective composite. Recently there have been examples of small scale Pepper’s Ghost models that create a “3D” display using a piece of glass or plexiglass and a large LCD/LED display position either on top of or below the angled reflective material.

This was a viable option because of the effect created however it offered a solution that was a faux 3D projection. If you were to look at the edges of the reflective material the effect is corrupted and the experience is ruined somewhat. It’s an interesting effect but in practice did not quite meet the standards we had set for the project. (See comparison to “Tabletop Systems” in subsection )

3.4.4 Decision

After very little deliberation we decided on the active shutter system because of the fact that Active 3D itself has many customizable variables and that we want to add our own head tracking system to the glasses. The entire 3D system is comprised of a transmitter for sending the timing of the image switching on the display and a receiver on the glasses for taking that signal and activating the “shuttering” between the lenses. We will need to decide whether we want to use a currently existing transmitter, however, it will be necessary for us to design our own active shutter glasses that includes the technology to allow for head tracking.

In the future, as glasses-free 3D technology advances and becomes more accessible from a viewer and a developer’s standpoint, this may become a superior option to 3D glasses technology because it creates an uninhibited
display for the user and does not introduce any additional necessities for use. Especially for our application, this type of technology would open a wide range of options for us to attack the other aspects of the project and reduce the complexity associated with 3D glasses technology. However, until then we must use what is most available and capable for what we want to do.

3.5 Hardware Requirements

In the following sections we are going to be discussing some of the major hardware decisions we made to most accurately match our initial design we had in mind. We will introduce the technologies we considered and explain the reasoning behind each of our individual choices.

3.5.1 Projector

The most important thing when choosing the correct projector for this project is being 3D capable. The second most is having a short enough throw to allow us to fit our projector in the self contained unit while also providing a high quality image. After looking at the market, most units either meet our needs perfectly however are prohibitively expensive or they slightly miss the mark but are completely within our price range. Majority of units on the market that have 3D capabilities, so finding one within our price range is not hard, but the best units for what we need are the ultra short or short throw projectors. The main issue with finding one of these projectors is they are extremely expensive for our budget. For that reason, we may have to shorten the throw of one of the more affordable projectors through the use of mirrors or lenses.

Projectors are separated into three different design types. These options are DLP, 3LCD, and LCoS and each has their own pros and cons.

3.5.1.1 DLP Projectors

More formally known as Digital Light Processing, DLP technology was developed by Texas Instruments in 1987. DLP Projectors are composed of tiny mirrors that correspond to individual pixels being projected onto the screen. These tiny mirrors turn “on” and “off” by either reflecting or not reflecting light toward the screen. The light shone alternately on the mirrors can be red, green, or blue and this light is decidedly reflected or not by these tiny mirrors which switch on and off depending on the video or graphics signal being fed into the underlying memory chip. These mirrors can switch at rates of thousands of times per second.

DLP Projectors vary largely in price depending on whether they are single-chip or 3-chip. The difference between the two types of projectors is in a single-chip projector there is only one chip that controls all the colors whereas in 3-chip
DLP there are three separate color chips to control the color of the pixels more accurately. DLP projectors can range anywhere from only a few hundred dollars up to tens of thousands of dollars depending on the model. Majority of the affordable 3D projectors utilize single-chip DLP technology. You can find models from Optoma, BenQ, Mitsubishi, and many others.

DLP projectors offer many positives when compared to the other projector types. DLP offers the ability to display both active and passive forms of 3D, provide completely jitter free images, and display a wider color gamut. One of the biggest draws for DLP is its easy maintenance and longevity. In these projectors you can easily replace the backlight if it ever goes out whereas with the other projector types you would normally need to have the light professionally changed. Another main factor that makes these extremely enticing is their lower average price for single-chip DLP projectors.

As with all things, DLP Projectors have one main drawback. The main drawback is that these tend to use more electricity and generate more heat than their counterparts. Another smaller drawback is there are sometimes issues with pixel color in single-chip projectors whereas competitors do not have this issue. Other than that, DLP offers an extremely affordable option with a large upside in comparison to comparable technologies.

3.5.1.2 3LCD Projectors

This now common form of LCD projection was first developed and refined by Epson and first used in projectors in 1988. 3LCD works by splitting white light from a lamp into red, green, and blue with the use of special dichroic filter/ reflector assemblies called dichroic mirrors. Each of these split beams of color then pass through their own individual LCD that sets each pixel at the correct brightness level. Once this is complete in each LCD, the beams are recombined in a dichroic prism that forms the final image which is then reflected out through the lens.

3LCD Projectors, much like DLP Projectors, feature the same price range of a couple hundred dollars up to thousands of dollars for pricier units. 3LCD also fall within our price range with some of their more affordable units. Examples of these projectors are available from companies like Epson, Panasonic, and others.

Much like DLP projectors, 3LCD projectors offer numerous positive aspects. 3LCD is known for having some of the best color light output especially when compared to single-chip DLP. The reason for that is because it contains three different LCDs in charge of controlling the colorization of pixels rather than a single source. These three LCDs also can create a finer image gradation making for a smoother final picture. A couple other big positives to 3LCD is these
projectors normally require less electricity to run when compared to other projectors of the same brightness rating, and when compared to 3-chip DLP are more affordable on average.

The disadvantages to 3LCD technology is limited. The main issue with this technology is the image quality normally is not as sharp as LCoS projectors. Other than that small issue people normally do not have any issue with 3LCD projectors. There are not many 3LCD options around the price range of single-chip DLP units, however they are comparable in price to 3-chip DLP units.

3.5.1.3 LCoS Projectors

More formally known as Liquid Crystal on Silicon, LCoS technology was first developed by General Electric in the late 1970s but was not fully utilized for large scale production until the early 2000s when Sony introduced the “Ruby” projector. LCoS works by using a CMOS chip to control the voltage on square reflective aluminium electrodes buried just below the chip surface, each controlling one pixel. Each plate responds to its own independent addressable voltage and adding or reducing voltage to that plate will cause it to appropriately increase or decrease reflectivity of the light.

LCoS projectors, on average, come at a much higher premium than both DLP and 3LCD. Introductory models start at prices in the thousands of dollars and top of the line models can reach the multi tens of thousands of dollars. Primary producers of LCoS projectors are Sony and JVC.

Between LCoS projectors and their competition, LCoS creates the superior picture quality. This all comes at a price though. They are prohibitively expensive in comparison to the other technologies and for that reason, even with their superior image quality, these projectors are at a large disadvantage compared to the other options.

3.5.1.4 Projector Decision

In the end, we decided on using a single-chip DLP projector for our project. This projector technology offered the best Price to Performance ratio. We do
not need anything with a magnificent picture quality, however we do need something that still has a sharp image and provides the ability to use active 3D. All the other options were just too expensive for the budget we had in mind for our project. There was nothing that fell within the price range we were looking for, but if we had a larger budget, utilizing one of those other technologies would only make it that much better.

If we were to use one of the other projector technologies we could potentially create an even better 3D experience for the user due to the increased frame rate and higher picture quality. An unlimited budget could have the most effect in this hardware decision because of the large ticket price of the superior systems. Although it is an example of the law of diminishing returns in regards to performance vs. price, there is definitely a noticeable difference between the lower and mid models and the top of the line models.

With one of the top of the line LCoS Projectors, not only could we create a more engaging experience from a picture quality standpoint, but also from a picture size view because of the capabilities of these projectors to display large images with exceedingly high quality pictures.

3.5.2 Infrared LEDs

A series of LEDs are needed as tracking points. When choosing the LEDs it is important we consider a number of factors. The first is the wavelength. Due to the choice of a 940nm filter on the camera and general availability, we will use 940nm LEDs. The choice of LEDs is then mainly influenced by viewing angle, maximum intensity, and power-efficiency. As follows, table 4.5 shows the top considerations of IR LEDs.

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Mounting Type</th>
<th>Forward Current (mA)</th>
<th>Radiant Intensity (mW/sr)</th>
<th>Voltage (V)</th>
<th>Relative Intensity at 45°</th>
<th>Radiance-Current Efficiency (μV/sr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APT1608F3C</td>
<td>Surface</td>
<td>50</td>
<td>2.5</td>
<td>1.2</td>
<td>80%</td>
<td>50</td>
</tr>
<tr>
<td>VSML3710-GS08</td>
<td>Surface</td>
<td>100</td>
<td>9.5</td>
<td>1.6</td>
<td>80%</td>
<td>95</td>
</tr>
<tr>
<td>VSMB2020X01</td>
<td>Surface</td>
<td>100</td>
<td>40</td>
<td>1.35</td>
<td>5%</td>
<td>400</td>
</tr>
<tr>
<td>VSMY3940X01</td>
<td>Surface</td>
<td>100</td>
<td>15</td>
<td>1.44</td>
<td>75%</td>
<td>150</td>
</tr>
<tr>
<td>VSMB3940X01</td>
<td>Surface</td>
<td>100</td>
<td>13</td>
<td>1.35</td>
<td>80%</td>
<td>130</td>
</tr>
<tr>
<td>LTE-R38381S-Z</td>
<td>Surface</td>
<td>1000</td>
<td>160</td>
<td>3.00</td>
<td>90%</td>
<td>160</td>
</tr>
</tbody>
</table>
### Table 4: IR LED Part Comparison

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Through-hole</th>
<th>100</th>
<th>600</th>
<th>1.65</th>
<th>&lt;5%</th>
<th>6000</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSLY5940</td>
<td>Through-hole</td>
<td>100</td>
<td>600</td>
<td>1.65</td>
<td>&lt;5%</td>
<td>6000</td>
</tr>
<tr>
<td>OED-EL-1L2</td>
<td>Through-hole</td>
<td>100</td>
<td>60</td>
<td>1.2</td>
<td>-- (30°)(^1)</td>
<td>60</td>
</tr>
<tr>
<td>EAILP05RDBA2</td>
<td>Through-hole</td>
<td>100</td>
<td>15</td>
<td>1.4</td>
<td>20%</td>
<td>150</td>
</tr>
<tr>
<td>MTE9440M3A</td>
<td>Through-hole</td>
<td>60</td>
<td>--</td>
<td>1.2</td>
<td>85%</td>
<td>--</td>
</tr>
</tbody>
</table>

It is worth explaining our consideration of surface mount parts verses through-hole. Through-hole parts give us the flexibility of placement within the glasses, however, they also take up more space when compared to the surface mount parts. The biggest drawback of through-hole parts is the lack of large-angle parts. That is, very few through-hole parts are designed to reasonably support viewing angles at 45° or more. Part no. VSLY5940 in table above shows this frustration, and most through-hole parts follow similar characteristics. Part no. EAILP05RDBA2 and OED-EL-1L2 being the best of the characteristics. Mentioning part no. MTE9440M3A which has potential to fit our needs, it has a high cost compared to other LED parts. Therefore, the high cost and lack of intensity information disfavors this part. In consequence, we are pushed to use surface-mount parts.

When considering each part, we then balance maximal intensity and viewing angle with power efficiency. Concerning voltage, our controller will supply greater than 3V, and any excess voltage can be optimally placed in an LED series circuit, or dampened through a resistor. Voltage is not a factor in part choice. As far as viewing angles are concerned, all remaining surface-mount parts with the exception of part no. VSMB2020X01 have a reasonable viewing angle. Part no. OED-EL-1L2 has a potential to fill our need, but it hard to be certain if the relative intensity at 45° is least 20% it will outperform other parts. However, to have a sustained intensity at most angles, it should be much higher (near 50%) so that the tracking system can sense near-uniform intensity. Given the nature of similar parts, this is unlikely. The top part of the remaining is therefore part no. VSMY3940X01. It has the highest intensity and efficiency of the parts with a reasonable viewing angle. We will then ending up using that part, and consider

\(^1\) Although there is no data for the intensity at 45°, this part has a viewing angle of 30°
part no. OED-EL-1L2 for further testing. If tests prove successful, the choice will come down to desire for through-hole or surface mount parts.

3.5.3 Transmitter/Receiver between Display and Glasses

Communication between our display, glasses, and the processing unit for head tracking is vital for our project to work at all. We have the option to use three main forms of communication, these being IR, DLP-Link, and RF Communication. In the following sections we will introduce each form of communication, their pros and cons, and what form of communication we have decided to choose.

3.5.3.1 IR Communication

IR communication uses light in the infrared spectrum to send signals between devices. Infrared radiation is characterized by wavelengths between 700nm and 1mm, just below the visible spectrum of light. Being that it is invisible to the naked eye, it makes IR a perfect option for communication.

The process of communicating through IR is not as simple as merely turning on a light indefinitely to send a message and turning it off when the message is done but it is close. IR communication is achieved through flashing IR LEDs at certain carrier frequencies and in specific patterns. The reason the specific LEDs require flashing rather than continuous on and off states is because these can require currents of up to 1A meaning more power usage and heat. By flashing the LED you still get the same signal strength just at a lower average power and you have the ability to customize the signal. These customizations are in the form of the carrier frequency, wavelength of light, and the IR code.

Carrier frequency corresponds to the frequency at which the IR light is modulated or flashed. Normally this value is between 30-60 kHz for commercial applications, the most prevalent frequency being 38 kHz. Along with the carrier frequency, wavelength of IR light can be specified for a more secure stream of communication between devices while eliminating a lot of external “noise”. Infrared radiation is everywhere in the world today from the sun to a lightbulb in your house, so making your stream of communication a certain wavelength of this spectrum of light naturally filters out a lot of these external sources. In most devices today, signals are sent at wavelengths between 800-1000 nm depending on the developer.

The final method of creating a unique IR signal for communication is using an IR code. An IR code is the pattern at which an IR signal at a specific wavelength and carrier frequency are sent. This pattern is defined through the use of the serial communication terms “marks” and “spaces”. “Spaces” are the periods in a signal where no light is being sent from the transmitter. During the “mark”
states of a signal the IR LED is pulsing at the desired carrier frequency. By varying the time of each of these periods and putting them in a specific order, IR codes can be developed to further differentiate the signal from others which for some can become frustrating because of the lack of a standard for IR codes. An example of this is use in televisions. Most televisions, even of the same brand, have different IR codes that correspond to their prospective remote and they do not work with other TVs that require a different code.

As all things, there are a couple drawbacks to using IR communication. IR requires a direct line of sight for a continued connection. If something crosses the path, connection would be lost and the timing of the glasses may falter. IR also requires a close range or again connection could be lost. Another negative aspect of IR is that because IR is still a form of light, there are many forms of noise that may affect the signal. However, if wavelength and frequency is specified it should not be much of an issue.

Infrared may be better for our application because implementation should be simpler than rf communication between devices. Our user should not be more than 9 feet of the transmitter and camera and there will not be anything blocking the path of communication so IR would be apt.

Most TVs and 3D projectors include 3D IR signal capability so all that would be necessary is receiving that signal and matching the glasses to it, however if we are prohibited by the cost of such a system we can get an active 3D capable system and use either an external 3D signal system or find a way to build our by matching the output signal with the frequency the tv is changing frames.

Vishay produces a number of 3D active shutter specific IR receiver ICs that we can use to further consolidate the size of our glasses PCB while maintaining the same amount of functionality as a larger system. Majority of these modules include the necessary photoresistor as well as amplifier and signal shaping steps for developing an easily readable signal for the MCU. Most of the Receivers offer a variable optimal carrier frequency for matching to a wide range of manufacturers because of the lack of a standard 3D shutter IR frequency.

There are several considerations then when it comes to part choice for an IR receiver. We wish to see the largest data rate, and thus the lowest latency for the system while having reliable transmission throughout the system. Several parts in table 3.4 are shown.

First note, that the high latency of part no. TSOP34156. This is due to the low data rate. We will be sending continuous data near 100 bits in length, and 10ms is far too large for the system since low latency is crucial to the quality of the head-tracking system. The IrDA parts offer a very high data rate on a reliable
technology. IrDA is a standard for IR communication. In unfortunately, has two drawbacks. The first is the range of the IrDA devices. Part no. TFDU4101 offers the best range, but only explicitly describes the range at a maximum of 1 m. Our system will need somewhere around 2m of reliable transmission distance. In addition to the range, IrDA devices only have an angle of half intensity at 24°. For head tracking purposes, this should be optimal at 45°. This unfortunate characteristic of IrDA devices suggests that IrDA will probably not fit our needs. However, further testing may prove otherwise. Finally, the remaining parts are more general and can transmit at data rates of 60 kHz. They also fit the need by having an intensity angle of 50 or 60°, and can transmit larger than 2m. The one drawback to part nos. TSMP6000 and TSMP77000 is the question of reliability. These parts are designed to be more general in the fact that they don't provide any modulation related to IR noise. Careful consideration must be met to avoid noisy IR environments to successfully have continuous transmission.

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Technology</th>
<th>Range (m)</th>
<th>Data rate (kb/s)</th>
<th>Latency @ 100 bits (ms)</th>
<th>Angle of Half Intensity (°)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSMP6000</td>
<td>Modulated IR</td>
<td>&gt; 2m</td>
<td>60</td>
<td>1.66</td>
<td>50</td>
</tr>
<tr>
<td>TSMP77000</td>
<td>Modulated IR</td>
<td>&gt; 2m</td>
<td>60</td>
<td>1.66</td>
<td>60</td>
</tr>
<tr>
<td>TFDU4101</td>
<td>IrDA (SIR)</td>
<td>0 to ≥ 1</td>
<td>115.2</td>
<td>0.868</td>
<td>24</td>
</tr>
<tr>
<td>TFBS4711</td>
<td>IrDA (SIR)</td>
<td>0 to ≥ 7</td>
<td>115.2</td>
<td>0.868</td>
<td>24</td>
</tr>
<tr>
<td>TSOP34156</td>
<td>Modulated IR</td>
<td>&gt; 2m</td>
<td>9600</td>
<td>10.8</td>
<td>45</td>
</tr>
</tbody>
</table>

Table 5: IR Receiver Parts

3.5.3.2 DLP-Link

DLP-Link is a form of communication created for the purpose of 3D Display and glasses syncing. Similar to IR, the glasses require a photodiode to read in the signal being sent from the display however because of the way the signal is sent no extra hardware is required and the things necessary for this communication can be programmed onto the existing hardware.

DLP-Link works by flashing a white light between each frame displayed on the tv. The glasses then match up their “blanking” state, or the state when both lenses are blacked out, with this white flash. Once the blanking and flashing are matched, the glasses are synced and simultaneously shutter eye to eye whenever each frame is displayed. There is only one standard for this communication type so all products on the market are interchangeable for all DLP-Link displays.
As with IR, there are a number of drawbacks to using DLP-Link for our communication method between our display and our glasses. DLP-Link is the same as IR in regards to the necessity of a direct line of sight to the display. Without a view of the screen the glasses cannot sync with the corresponding white flashes. Also much like IR communication, the range of the DLP-Link connection is not very good but good enough. Unwanted noise can also be an issue with DLP-Link connections as well.

As of now we're still researching what type of photodiode receiver is necessary for DLP-Link connections. There are no published articles defining the characteristics of the white flash or how DLP-Link technically works. At the moment we're trying to reverse engineer a current pair of glasses so we can try to figure out the signal frequency and wavelength so we can mimic that. If anything we may have to use one of the receivers from an existing pair of glasses to pick up the correct signal consistently.

3.5.3.3 RF Communication

RF communication is a means of transferring data through forms of light other than visible or infrared, normally in the radio to microwave range. There are many standards and signal forms in RF communication, however in the scope of 3D glasses communication the main signals are dependent on the manufacturer. The signal is sent out from the display and the signal is picked up and read from the glasses.

For this to be done a transmitter in the display and a receiver in the glasses will be necessary. Unlike the other two forms of communication, a single photo-diode and simple filter will not cut it as a receiver. The glasses would require an antenna for receiving the signal sent from the display as well as a filter to demodulate this signal. In terms of hardware, there is not that much difference from the setup for IR or DLP-Link communication, however more work will have to be done by the MCU to filter and read the signal. It is not overly complex but it does add some complexity to the glasses where space is limited.

RF communication is a very attractive option because it allows the glasses to remain connected to the display even if line of sight is broken. Overall, it provides a stronger connection to the display so even under heavy movement or rotation the connection will not break. The caveat is that this form of communication is a bit more expensive than a simple photodiode setup and may take more space then we have to offer on the PCB.
3.5.2.4 Communication Decision

<table>
<thead>
<tr>
<th>Communication</th>
<th>Line of Sight</th>
<th>Extra Hardware</th>
<th>Standard of Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>Yes</td>
<td>Yes (Transmitter)</td>
<td>No (Proprietary Coding)</td>
</tr>
<tr>
<td>DLP-Link</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RF</td>
<td>No</td>
<td>Yes (Trans and Filter)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 6: Communication comparison

After weighing each of our options and collaborating with one another to determine what would be best, we have decided to use DLP-Link for communication between the display and the glasses and IR for communication between the glasses and the processing unit/camera.

DLP-Link offers the easiest solution for communication between our glasses and the display because of its standardization and widespread availability. When compared with IR and RF, it provides nearly the same benefits without the need for extra hardware or guessing the coded frequency of the sync signal. On the glasses side it requires the same amount of hardware as using IR and less hardware than would be required for us to utilize RF. Overall, it is our best option for display to glasses communication.

For the transfer of data from the glasses to the processing unit we chose IR communication because of its simplicity and the availability of current software that we can utilize to interpret the data to use for head tracking. RF would require a higher amount of power from the glasses and more hardware that would take up precious space on our PCB for the glasses. Although RF was an option, it was not feasible for our project so IR would be the best choice.

3.5.4 Camera System/Data Transmission

As far as the camera system, IR would work very well also because we can use IR LEDS and an IR Camera to read in the position and orientation of the glasses for head tracking. There are many examples of this implementation using the Wii remote camera and IR LEDS fixed on a pair of glasses or a headset.

For further accuracy, we are planning to also use data from a gyroscope and accelerometer. There are a couple methods of transferring this data, however we’ve decided bluetooth would be the easiest solution for our needs.

3.5.4.1 Bluetooth

Bluetooth is a good choice for a wireless transfer protocol because it is ubiquitous and widely supported. The new Bluetooth Low-Energy (BLE) is a
significant protocol designed for smaller devices and uses less energy. BLE is
designed for small chunks of data, which is what we will be doing with it.
Devices that use BLE incorporate the Bluetooth Core Specification Version 4.0
(or higher). The range for BLE is 50-150m and the data rate is 1 Mbps.

BLE is optimized for low-latency sporadic transmissions however, and its
efficiency degrades for larger data transfers. It has a maximum of 20 byte
payload per packet. With a gross 1 Mbps data rate, we have a theoretical
maximum transfer rate of 250 kbps, and in practice the actual transfer rates
drops below 100 kbps. In practice the actual transfer rate drops below 100
kbps. Theoretically we should be able to get away with sending X B of data 72
times a second (72 fps).

\[
\text{bitrate (bps)} = \text{packet size} \times \text{transfer frequency} \\
\text{packet size} = \text{gyro data} + \text{accelerometer data} = + 2B \times 3 + 2B \times 3 = 12B \\
\text{transfer frequency} = 72 \text{ Hz} \\
\text{bitrate (bps)} = 12B \times 8b/B \times 72 \text{ Hz} = 6.91 \text{ kbps}
\]

We decided to not go with Bluetooth in lieu of using IR transmission of data
because of the lag present when using bluetooth devices. Bluetooth was not
designed around responsiveness so we’d rather write something that is
transmitted at the speed of light that works at the hardware level.

3.5.6 Sensors

Apart from our decision to use the IR Leds to determine the position of the
user’s head in relation to the display, we are also planning on utilizing other
sensors to accurately read each of their individual movements rotationally and
in the z-plane. The addition of these extra sensors can eliminate any lag that we
may experience from using only the IR camera and can add extra accuracy to
the movements that we read in from the user. The main additional sensors we
are considering are a gyroscope and an accelerometer.

3.5.6.1 Gyroscope

Gyroscopes are devices which measure angular velocity in three dimensions.
Angular velocity \( \omega \) is defined by the following equation:

\[
\omega = \frac{d\theta}{dt}
\]

Gyroscopes are particularly useful when trying to measure movement in three
dimensions. If we put one on our glasses we could more precisely track 3d
movement.
Gyroscopes work by putting a mass on a spring and when it moves out of position it emits an electric signal that can be amplified.

For our purposes we will only be interested in rotation on the X axis (the axis which lies on the horizontal line between the user’s’ eyes) because we can determine the x,y,z position as well as roll and yaw from our IR sensors.

One major benefit of the use of a gyroscope is it will give us the best rotational data with the least lag from the users movement to the gaming display. If we were to simply rely on the movement of the IR LEDs read in by the camera, the system would be dependent on the flickering of the IR LEDs, the ability for the camera to process the movements, and the ability of the head tracking software to transfer that movement into movement in the game. By using the gyroscope we eliminate the steps associated with pieces of hardware being made to do something they’re not specifically intended to do. In the gyroscope system, everything functions as it was intended. The gyroscope reads in the movements and that data is transferred directly to the software to use and apply to the game. All the movements are already interpreted and nothing needs to be processed in the software.

3.5.6.2 Accelerometer

Acceleration $a$ is defined by the following equations:

$$
\frac{da}{dt} = \frac{d^2 x}{dt^2} \\
F = ma \Rightarrow a = \frac{F}{m}
$$

An Accelerometer is an electromechanical device that sense static (such as gravity) or dynamic (such as movement) forces of acceleration. Once again, putting an accelerometer on our glasses can help us more precisely track the movement of our users’ heads.

Accelerometers work by placing one plate of a capacitor on a mass. Changes in acceleration affect the position of the mass, thus changing the capacitance, which can be measured electrically as a voltage.

Similar to the gyroscope, the use of an additional accelerometer to determine the movements of the user should reduce the amount of lag in the system. Everything again would be designed to function as it was intended, thus eliminating a lot of the processing that would be necessary with only the IR LED system.
3.5.7 LC Lenses

We are planning on using lenses off a pair of active 3D glasses that we buy. We will need to determine voltage necessary to make the lense opaque. The Voltage values that normally correspond to the best coverage range anywhere from 10 V to 20 V. Depending on necessary boost voltage we need a corresponding resistor value to correctly supply that voltage.

Another major idea associated with the lense is the transparency when not activated. Lens transparency can range anywhere from 20% to 40% depending on the quality of the lens, the greater the value the better. A higher percentage equates to a larger amount of light being allowed through the lense and a more uninhibited viewing experience.

3.5.8 Battery

The most critical factors that normally go into choosing a battery are both the Voltage and the batteries capacity in Ah. Because the circuit we’re going to be running needs to be as small as possible, size is also an important aspect for us to consider. All options on the market today use either rechargeable or nonrechargeable batteries, each having their own positives and negatives.

3.5.8.1 Rechargeable

Obvious benefit of rechargeable batteries is the fact that you do not have to go out and replace the battery whenever it dies. All that is necessary is a quick plug into the charger and you are good for another battery life. The main issue with rechargeable batteries are that they do not have the same storage capacity as comparable sized nonrechargeable models. We will need to find a battery that is still minimal in size but capable of holding a charge for a reasonable amount of time.

Two rechargeable battery models we are considering are the LR2025 and LR2032 3.6V. Much like their non-rechargeable counterparts they offer a very good size to storage ratio. The LR2025 has a capacity of 40 mAh and the LR2032 has a capacity of 50 mAh meaning they will have only about a third or fourth of the runtime of their nonrechargeable counterparts. If you were to equate this to hours, current models on the market using the CR2025 can run for about 150 hours at a time at a capacity of 150 mAh so we would be looking at 40 hours at a time for the same size rechargeable battery if you were to run it continuously. Because the PMIC unit we are considering only allows a recharge rate of 100 mAh it will take about half that time to recharge completely, however this can be done whenever the glasses are not in use. We are also considering offering the ability to continue to use the glasses when wired.
3.5.8.2 Non-Rechargeable

Non-rechargeable Batteries offer a good option that allows for a longer battery life without the need for replacement or recharging, however the biggest Issue with non-rechargeable batteries is that you will have to physically replace them at some point. This can be an inconvenience, especially if the battery only lasts for a short amount of time and the user needs to replace it often. We’re looking for a battery that meets a nice balance between storage capacity and size.

The two models we are mainly considering are the CR2025 and CR2032 3V. These batteries are the most commonly used with current 3D glasses. These are perfect batteries for our purpose because they are of sufficient voltage to power the PMU and shutter system at a good storage to run for a long time. At 3V the CR2025 has a capacity of around 162 mAh and the CR2032 has a capacity of 225 mAh but the 2025 is only 2.5 mm compared to the 3.2 mm 2032. Depending on the capacity constraint of our battery we may be able to go with the 2032 for longer amounts of time between charges, however if size becomes an issue, the 2025 would be more appropriate.

If worst comes to worst, we have the option of using the CR2016 3V which features an even shallower depth of 1.6mm, but at 90 mAh the runtime will be drastically reduced compared to the other batteries.

3.5.8.3 Battery Decision

<table>
<thead>
<tr>
<th>Model #</th>
<th>Storage (mAh)</th>
<th>Depth (mm)</th>
<th>Estimated Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR2016</td>
<td>90</td>
<td>1.6</td>
<td>90 hours</td>
</tr>
<tr>
<td>CR2025</td>
<td>162</td>
<td>2.5</td>
<td>162 hours</td>
</tr>
<tr>
<td>CR2032</td>
<td>225</td>
<td>3.2</td>
<td>225 hours</td>
</tr>
<tr>
<td>LR2025*</td>
<td>40</td>
<td>2.5</td>
<td>40 hours</td>
</tr>
<tr>
<td>LR2032*</td>
<td>50</td>
<td>3.2</td>
<td>50 hours</td>
</tr>
</tbody>
</table>

* Denotes Rechargeable Battery

Table 7: Battery part comparison

The chart above lists the battery options we are considering for our device. Although there were many other available batteries, we were able to narrow down our search to these five. Our final choice for our 3D Glasses circuit was the CR2032. The CR2032 offered the best amount of storage for the size of the package. With the smaller depth we have more room to add other necessary components we need while also maintaining a good battery life without the need for constant recharging. For example, the non rechargeable version of the CR2032 has almost 5 times more battery life of its rechargeable counterpart.
3.5.9 Power Supply Circuit

![Power Supply Circuit Image]

**Figure 3**: Example of TPS65835 in a pair of 3D glasses

We must develop a method of powering the glasses with a battery either rechargeable or not. To develop the correct power system we need to estimate how much voltage and current we will need to power the glasses. After researching other examples it looks like we will need anywhere from 3 to 5 volts of power and use about 1 mA of current. Within the circuit we will need some sort of voltage regulator and a boost converter to power the higher voltage activate liquid crystal lenses.

For the power supply circuit we are considering using either the TPS65735 or the TPS65835 Power Management Units designed by Texas Instruments. These PMUs are specifically designed for use in 3D active shutter glasses applications and the only difference between them is the 835 model also contains a built in MSP430. The PMU IC consists of a three phase linear charger, an MCU low dropout regulator, DC-DC Boost Converter, and full H-Bridge analog switches for the left and right shutter control in the glasses, all in a tiny VSON(32) 4mm² package. This small package size is critical to the development of the board on the frames of the glasses and give us plenty of room for other necessary parts.

The PMU will be responsible for also regulating the recharging of the battery and dispersing the necessary power to all the components of the glasses including the MCU, LC Lenses, and the DLP receiver. We're considering powering the IR LEDs separately due to the high power draw. We would be able to maximize these separate batteries lives using MOSFETs to modulate the IR LEDs output.

3.5.10 MCU

Mainly in charge of signal for regulating and matching image frequency with LCL shuttering frequency. The MCU will receive either a demodulated IR or RF signal and will have to use it to power on and off the left and right lens in sync with
the display unit. The unit must have a low power sleep mode to limit power usage if left on but not in use. There are a few options out there specifically made for 3D glasses technology however it is not a necessity depending on the other units in our circuit.

The three MCUs we are planning on using for our project are the EFM32 Gecko by Silicon Labs, the MSP430 by Texas Instruments, or the Holtek HT45FH3T. Most of the models on the market utilize one of these three MCUs and in the following sections we will go over the capabilities of each and our decision between the three.

3.5.10.1 EFM32 Gecko

The EFM32 MCU utilizes a ARM Cortex M3 core and is marketed specifically for systems and products requiring high performance at a low energy consumption. The 32-bit MCU has a CPU that operates at speeds of up to 32 MHz and features a short wake-up time from its energy savings modes and it is capable of five different power modes. With memory options reaching 128 Kb of flash memory and 16 Kb of RAM, and IO numbers reaching up to 90 pins with 20 mA drive strength, the EFM32 offers a very powerful MCU that is more than suitable for our needs.

Overall, this MCU fits our needs really well. It offers the necessary computing power while also offering different sizing for us to choose from for our compact PCB. With the addition of the five power modes, compatibility with UART, USART, and SPI, inclusion of an ultra low power ADC and DAC at 1 Msps and 500 Ksps sampling rates, and a low input voltage of 1.98-3.8 V the EFM32 would fit into our design well. All these offerings can be used to maximize the efficiency of our glasses design.

The main drawbacks of this MCU have less to do with the technical specifications associated with it and more to do with fitting all the necessary components of our design onto a PCB small enough to fit into a pair of glasses comfortably. By using this MCU we would need the corresponding resistors and capacitors necessary for it to work as well adding a lot more surface area and complexity to our PCB design. Along with that, compatibility between this MCU and our chosen PMU is not a given. We may have to go through a much longer troubleshooting phase to make sure all portions of the MCU and the PMU, which controls the actual shuttering, match up and communicate correctly.

3.5.10.2 MSP430

The MSP430 is a tried and true MCU that we were all familiar with, however it is no slouch in the technical specifications department. This 16-bit MCU boasts a CPU that can operate at a speed of up to 16 MHz and just like the EFM32 offers low power operation with five different power saving modes. Although it only
has 16 Kb of flash memory available, for our needs we should not need anything more.

This MCU is pretty much specially designed for applications like our 3D glasses idea. With features like UART, SPI, and I2C compatibility, a 10-bit ADC that samples at a rate of 200 Ksps, and Serial Onboard Programming, there is not much more we would need from an MCU. However the MSP430 further sets itself apart from the competition.

The TPS65835 is one of the power management units we were looking at using and unlike the TPS65735 it also includes a built in MSP430 core specifically for use in 3d glasses systems. Having our MCU combined with our PMU component eliminates a lot of the surface area that would be necessary if we were to use separate PMU and MCU ICs. This downsize also would not be limited to just a decrease from two ICs to one IC but we would also lose the need for the larger number of ancillary capacitors and resistors. In reality this will save us a lot of surface area on our PCB making the idea that we will fit everything more realizable.

3.5.10.3 Holtek HT45FH3T

The Holtek HT45FH3T is a very interesting option as an MCU for the 3D glasses. As strictly an MCU this 8-bit unit contains a CPU capable of running at speeds of up to 4 MHz, but do not let those low numbers fool you. With an included 12-bit ADC, 10 and 16-bit periodic timers, and two additional 10-bit Compact Timer Modules this little MCU is built for the purpose of operating a pair of 3D glasses. Also just as the MSP430 comes included in the TPS65835 PMU, the HT45FH3T boasts similar added features like an included 10V boost converter for shuttering the Liquid Crystal Lenses, four level shifters for controlling that boost converter, and extra power management features for managing battery life and charging.

In all, this MCU is a great option for designing a pair of 3D glasses. It offers many of the same features that both the MSP430 and the EFM32 offer with the additional options that would come in the TPS65835 PMU/MCU IC.

3.5.10.4 MCU Decision

<table>
<thead>
<tr>
<th>MCU</th>
<th>Manfctr</th>
<th>CPU (MHz)</th>
<th>Flash (Kb)</th>
<th>Low Power Modes</th>
<th>ADC (bit/sps)</th>
<th>PMU included</th>
<th>Boost Convert</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFM32 Gecko</td>
<td>Silicon Labs</td>
<td>32</td>
<td>128</td>
<td>5</td>
<td>12/1M</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

37
<table>
<thead>
<tr>
<th>MSP430</th>
<th>T.I.</th>
<th>16</th>
<th>16</th>
<th>5</th>
<th>10/200 K</th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HT45FH3T</td>
<td>Holtek</td>
<td>4</td>
<td>32</td>
<td>2</td>
<td>12/500 K</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Table 8: MCU Part Comparison**

After going back and forth between our options we decided to go with the MSP430 in the TPS65835 IC. This MCU offered the best compromise between power, size, and functionality. The decision between this and the Holtek HT45FH3T were extremely close however the added features of the TI PMU/MCU duo beat out the Holtek model. The Universal Serial Communications Interface and timing modules that support easy synchronization with DLP-Link communication made the MSP430 PMU IC stand out just enough.

![Figure 4: Size comparison of TPS65835 PMU/MCU with a U.S Dime](image)

As far as the EFM32 was concerned, it looks like an extremely capable option for 3D glasses technology but it introduced too much complexity because of the amount of PCB space that would be taken by the MCU and PMU alone. For that reason we decided to pass on that MCU.

**3.5.11 IR Transmitter**

Based on other implementations we saw online, we are planning on simply using IR LEDs that will be connected to our glasses to send out the position and orientation of the user. No signal will be sent just the IR light and the camera will determine position based on its relative location. The IR LEDs are going to be powered separately from the PMU because of their higher power draw. We will integrate a MOSFET switching circuit to modulate the IR LEDs so as to limit
the amount of power being drawn.

3.5.12 DLP-Link Receiver

The DLP-Link receiver is one of the more important parts in our 3D glasses design. It is responsible for receiving any and all of the information so the glasses can sync with the picture on the display. When we were looking for optical sensors that may be appropriate for receiving these visible white light we were not able to find very much information at all. A lot of the information on DLP-Link is kept secret so we decided at the moment it may be best to just use one of the light sensors from a working pair of DLP-Link 3D glasses. Until we can determine the best option otherwise, using a sensor from an existing pair is our best option.

![Figure 5: Size comparison of the DLP-Link Receiver with a U.S Dime](image)

3.5.12 PCB

All the major electrical components will be mounted to a printed circuit board (PCB). The PCB will be responsible for housing the processing components as well as providing the capability to handle all the power distribution and regulation from the battery. We will have all sorts of hardware components such as the microcontroller (MCU) as well as the IR transmitter/receiver which we will go into detail about later. These parts will need to be soldered on to ensure them from moving and we will then be able to begin communication from the transmitting and camera unit to the glasses. This is the centerpiece of our project.
3.6 Tracking System

3.6.1 Raspberry Pi

Raspberry Pi is a full fledged computer that is very small and cheap. It uses 32 bit ARMv6Z architecture, a single core 1GHz ARM1178JZF-S Processor, 512 MB of SDRAM. A MIPI CSI (Mobile Industry Processor Interface Camera Serial Interface), a bluetooth radio, an unpopulated 40-pin GPIO header, and an unpopulated composite video header.

The Raspberry Pi will serve as our means of collecting IR triangulation, accelerometer and gyroscope data via bluetooth or an IR camera. Bluetooth is built into the Raspberry Pi as well as the MIPI camera interface, so the RPi gives us some options.

3.6.2 Raspbian

Raspbian is the Raspberry Pi’s Debian based operating system. Raspbian is highly optimized for for the Raspberry Pi’s low-performance ARM CPU's.

3.6.3 USB On-The-Go

We can have the Raspberry Pi act as a USB device using Universal Serial Bus On-The-Go (USB OTG). USB OTG is a way of getting the RPi to act as a usb slave instead of a USB master. Since USB can only be configured in a master-slave relationship, and since the raspberry pi is itself a computer (and by default a USB controller), this feature is crucial.

This allows us to access several useful modules, such as Serial and Ethernet modules. We can send over triangulation, gyroscope and accelerometer data (read from wireless inputs) to any listening device. Our software would loop and grab any available data from the input buffers, package it accordingly, and send it to the USB serial buffer.

We would output to USB put writing to a serial port. A simple way we can do this is by writing a python script and utilizing the serial.py package.

3.7 Rendering System - Windows

3.7.1 Reading from the Raspberry Pi

We will need to use windows to accept the output of the Raspberry Pi Zero and for unity to be able to use it. The Raspberry Pi Zero will be treated as a USB
device and must be opened and read from as such, as Windows would see no difference between the Pi and an actual USB device. To achieve this we can write a C program and make use of the Windows Win32 API.

The **Win32 API** is a library that can be imported to a C program that is running on Windows. It has support for the following functions we will need to utilize in order to read from our Raspberry Pi Zero:

**CreateFile()**

```c
HANDLE WINAPI CreateFile(
    _In_   LPCTSTR lpFileName,
    _In_   DWORD dwDesiredAccess,
    _In_   DWORD dwShareMode,
    _In_opt_   LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    _In_   DWORD dwCreationDisposition,
    _In_   DWORD dwFlagsAndAttributes,
    _In_opt_   HANDLE hTemplateFile
);  
```

This makes the initial connection to our USB device. lpFileName will be something along the lines of “COMX” (where X will be an integer 1,2,3…) which will be the name of the USB port we are reading off of. We will have to check the output of CreateFile() for errors to make sure that the port name we specified was valid.

**SetCommState()**

```c
BOOL WINAPI SetCommState(
    _In_   HANDLE hFile,
    _In_   LPDCB lpDCB
);  
```

SetCommState() is what we will need to use to configure BaudRate, ByteSize, StopBits, and Parity settings for our USB reader.

**SetCommTimeouts()**

```c
BOOL WINAPI SetCommTimeouts(
    _In_   HANDLE hFile,
    _In_   LPCOMMTIMEOUTS lpCommTimeouts
);  
```

SetCommTimeouts() is used to configure timeout settings for our USB reader.
SetCommMask()
BOOL WINAPI SetCommMask(
    _In_   HANDLE hFile,
    _In_   DWORD dwEvtMask
);

SetCommMask() is what tells our USB device to listen for a certain type of event. dwEvtMask is an type that determines the type of event that has been passed through.

Other Functions
Other functions that are worth mentioning but do not need too much exposition are WaitCommEvent(), which stops execution until a communication event has been read, and CloseHandle(), which closes a communication channel.

3.7.2 Working with DLL’s
In order for our Render Engine to see the information we have to interface with it using a .dll file. DLL is a filetype in windows and stands for Dynamic Link Library. This filetype was first introduced with the first releases of Microsoft Windows, and is a fundamental structural component of the OS. DLLs allow certain code fragments to be compiled into a single library, and to be linked to by multiple programs.

__declspec()
We can use the __declspec keyword to identify which functions you wish to make available to other executables. We will look at two __declspec identifiers.

- __declspec(dllexport) - used for exporting dll's to be made available to other programs
- __declspec(dllimport) - used for importing dll's that are declared in other programs.

When writing a dll, we need to use the dll export keyword to denote functions that are going to be available to other programs. We will have to include this __declspec(dllexport) at the beginning of any function prototype who’s defining function will be accessed by other programs.

__declspec(dllimport) is used to declare function prototypes that are pulled in from other methods.

A universal macro which is used in header file creation is as follows:

```c
#ifdef BUILDING_DLL
#define DLL_FUNCTION __declspec(dllexport)
#endif
```
#else
#define DLL_FUNCTION __declspec(dllexport)
#endif

This takes care off all our header declarations, in one fell swoop, we will not need to write separate files or prototypes to handle importing and exporting the function prototypes.

**DLLMain**

Every dll needs a DllMain function, it is what windows calls to link a DLL to a program. It is defined as such:

```c
BOOL APIENTRY DllMain (HINSTANCE hInstance, DWORD reason, LPVOID reserved)
```

The DllMain function must return true if the library was loaded successfully, or false otherwise.

**Static and Dynamic Loading**

DLL libraries can be linked to an executable in two different ways: Statistically and Dynamically. When statically linking to a DLL, the linker does all the work. It will be obvious from code namespace that the functions are located in an external library.

Dynamic DLL linking happens at execution time, while the program is running without the need to be recompiled. This is possible by calling the LoadLibrary function from the **Win32 API**. The prototype of LoadLibrary is

```c
HMODULE WINAPI LoadLibrary(LPCTSTR lpFileName);
```

lpFileName is the name of the dll you are trying to include. The OS will check your PATH for that dll and if successful will load it.
3.8 Error Checking

3.7.3 Data sent from Hardware to Unity

The Infrared cameras we plan to use to pick up the beams of Infrared light outputted by small emitters that will be placed on the viewing aperture will serve no other purpose than to determine the exact height the object is, what its x, y, and z coordinates are compared to the calibrated camera x, y and z coordinates. The project will either consist of one camera, or two cameras to track all three variables.

While we will have access to the exact position of the object in question, the software portion of the project is not concerned with this much detail and needs it to be boiled down to only the essentials in order to maximize the frame rate limit on the desktop PC running the Unity Engine. This is where our board comes in, which will receive either one or two sets of x, y and z coordinates and calculate from them the precise “angle of attack” that the viewing aperture is from “center line” which is the straight line perpendicular to the flat surface of the screen upon which the image or images will be projected.

Once this calculation is done, using simple trigonometry, the angle of attack (positive or negative) will be converted using the JSON Serialization utility we discussed a page earlier in order to send that data over to the Unity Engine. This
is where our project is separated from traditional three dimensional viewing apertures.

Most attempts at 3-D in the past few decades have been much more similar to two and a half D. An excellent example of this is the first shooter recorded, Doom, which was not a fully three dimensional interactive experience but rather two dimensional sprites which always faced the player.

As shown, the sprites didn’t actually have depth, when you moved closer the sprites would only become larger to imply distance. The only actual 3D portions of this interactive experience were the level design itself. Which is exactly how old “blue and red” 3D glasses worked, it wasn’t 3D it just had linear depth, and the depth wouldn't change as your perspective changed unless it was along the line of depth directly to and from the center of the screen.

However, our project will send this “angle of attack” value to the computer which will then move the virtual cameras to compensate, thus giving the same angle of attack to the viewer.

### 3.8.1 Hamming Code

Hamming Code is one example of a simple CRC method that can be used. It makes use of parity bits, which are bits which can describe whether the amount of 1’s (or 0’s) present in a subset of a bit string are even (or odd). The type of parity (and, in fact, the method of CRC) is something that must be agreed upon by both the transmitter and receiver beforehand.

The number of parity bits which are decided upon have to cover all of the bits in the bit string. Parity bits are placed at positions matching up with the powers of 2 (1,2,4,8,16...) and cover successive alternating bits corresponding to what that power of two equals. (Parity bit 1 covers every other bit, parity bit 2 covers every other two bits...)

When parity bits are set they can be used by the transmitter to check up to two incorrect bits and fix a single incorrect bit. For a single fix, the position of an incorrect bit can be determined by summing the positions of the parity bits which are incorrect. An incorrect parity bit is determined the same way that they are set, by looking at successive alternating bits which alternate every n bits, where n is the position of the parity bit.

### 3.8.2 Cyclic Redundancy Checks

A cyclic redundancy check is a check that is done to a packet or string of data that comes into a receiver. It is a method of checking that packet or string of data for errors that may have occurred due to a faulty or potentially error prone
transmission material. It is important to have something which checks for these errors to prevent any possible miscommunications so that data can be retransmitted.

An example of such an error is a radio wave that has been transmitted through a medium that is not conducive to radio transmissions such as lead. This would cause the radio waves to be dampened inconsistently (if the transmitter was passing through inconsistent layers of lead) which may cause the receiver to think there may be a high where a low is truly present and a low when a high is truly present.

It is important to have a method which checks for these errors to prevent any possible miscommunications so that data can be retransmitted if necessary so that bad information is not used. First we must go into a mathematical background to understand what kind of Cyclic Redundancy Checks exist and how Cyclic Redundancy can be done.

### 3.8.3 Checksum

A prominent method of Cyclic Redundancy Checking is called an Internet Checksum, or checksum for short. It is based on the mathematics that a number $n$ composed of $m$ digits in base $b$ can be represented as follows:

$$n = a_m \times b^m + a_{m-1} \times b^{m-1} + \ldots + a_0 \times b^0$$

This is also the format for polynomials. When the polynomial is divided by another polynomial which both sides of transmission agree on called the generator polynomial it yields a remainder. That remainder is added onto the end of the transmitted bit string and the same division process is repeated. This time, a remainder of zero is expected and if a zero is not returned by the division process that means there was an error produced by the transmission media.

### 3.8.4 Cyclic Redundancy in USB

CRCs are useful because they are capable of detecting all single and double errors and many multiple errors with a small number of bits. Communications protocols often use two CRCs in a packet - one to protect the header of the packet and another to protect the data portion of the packet. In USB the header of the packet is the PID field. Since this field is only 4 bits long it is protected by a 4 bit check field derived by simple bitwise inversion of the PID field. This provides adequate single-bit and burst error protection without requiring CRC generation and checking logic. The 'data' portion of a USB packet which is longer is protected by a conventional CRC field. In token packets, the CRC protected region is only 11 bits - so a 5 bit CRC provides adequate protection and also
aligns the packet to a byte boundary. Data packets may have up to 1023 bytes; so a longer (16 bit) CRC is used to protect the data.

The USB spec lists two generator polynomials - one for tokens and the other for data packets. The generator polynomial for tokens is \( x^5 + x^2 + x^0 \) while the generator polynomial for data packets is \( x^{16} + x^{15} + x^2 + x^0 \). Since the remainder is always of smaller degree than the generator polynomial, the token CRC is a 5 bit pattern and the data CRC is a 16 bit pattern.

### 3.8.4.1 CRC-16 Implementation

The implementation of CRC-16 on the packet header is defined below:

- The polynomial for CRC-16 shall be 100Bh.
- Note: The CRC-16 polynomial is not the same as the one used for USB 2.0.
- The initial value of CRC-16 shall be FFFFh.
- CRC-16 shall be calculated for all 12 bytes of the header information, not inclusive of any packet framing symbols.
- CRC-16 calculation shall begin at byte 0, bit 0 and continue to bit 7 of each of the 12 bytes.
- The remainder of CRC-16 shall be complemented.
- The residual of CRC-16 shall be F6AAh.

- Note: The inversion of the CRC-16 remainder adds an offset of FFFFh that will create a constant CRC-16 residual of F6AAh at the receiver side.

### 3.8.4.2 CRC-5 Implementation

- The CRC-5 polynomial shall be 00101b.
- The initial value for the CRC-5 shall be 11111b.
- CRC-5 is calculated for the remaining 11 bits of the Link Control Word.
- CRC-5 calculation shall begin at bit 0 and proceed to bit 10.
- The remainder of CRC-5 shall be complemented, with the MSb mapped to bit 11, the next MSb mapped to bit 12, and so on, until the LSb mapped to bit 15 of the Link Control Word.
- The residual of CRC-5 shall be 01100b.

Note: The inversion of the CRC-5 remainder adds an offset of 11111b that will create a constant CRC-5 residual of 01100b at the receiver side.

### 3.9 Application Design System

The difficulties between choosing a particular gaming engine delve into three basic topics: Ability to transfer the visual to a stereoscopic three dimensional format, the ability to cooperate with other input and output devices and the
ability to run at 30 frames minimum at all times. According to both the Unreal Engine and the Unity Engine's web pages, neither software comes prepackaged with the software to output stereoscopic 3D in their free versions. Unity does have it packaged with its Prop version, but that is outside of the project's budget. Despite this, there exists third-party software that enables Unity to have this functionality provided free of charge in the community workshop. Unreal Engine does have a 3D functionality prepackaged in any of its free or paid versions, however the implementation is very shoddy and renders many different textures and shapes incorrectly, making it a non-option for our purposes.

Investigating the third-party software yielded mixed results as the performance could not be determined and the projector had not come in for testing. However after discussing in detail with the team members about this dilemma an alternative option was presented. This idea was the Nvidia graphics card 3-D output format. Which was readily available as this card was near-necessary in order to run high quality three dimensional simulations.

After all this was settled the decision between the two engines was still not decided. This meant we needed to develop some new criteria for the comparison of these two engines. Another engine could have been chosen however after research it seemed that these two had the most features and the most platform compatibility. A member of our team had personal experience with Unity as they had created a few projects beforehand with a different team, who I then consulted. This resulted in a few suggested category of focus and they are as follows.

1. **Camera control in order to account for viewing position of a head tracked user.**
2. **Efficiency of running multiple instances of itself on one processor and graphics card.**
3. **Modifiable parsing of non-standard input via USB port and UART or ASCII processing.**
4. **Ease of sharing and updating game versions during development process.**

After these were considered, Unity engine was the obvious choice, as we had delved into the camera control as well as run multiple instances during my multiplayer testing. Additional experience had been done database parsing inside of unity through online sources so that made it functional. Also, according to Unity’s website all users are permitted to use something called the Collab system, which allows multiple users to share updated versions of the project and make comments or notes on what was changed and why, making the ease of development a hugely attractive factor. Thus Unity would be the engine to run our holographic projections.
3.9.1 Viability of other Gaming Engines

Instead of the Unity Engine, could we use a different game engine, either free or paid to use? While it would be possible to create a three dimensional interactive experience completely out of nothing but JavaScript, it would be a nightmare in terms of time. The reason interactive experience developers use game engines already made instead of creating them from scratch is because the creation of a solid game engine can take years to do and costs the company millions of dollars. Game engines provide support for the physics engine, or how a block will fall, how fast it will fall, how it will tip over when placed at the edge of another block, how walls are formed, how the repositioning system makes it so that a player or physical game object cannot pass through the wall, how walls connect, how the small space between them cannot be slipped through and many many other premade useful little pieces but one stands out much more than the others. The camera itself, and how it interacts with the environment. A game engine has to know when looking at say a set of blocks, what blocks are visible? What blocks are behind walls? (This is called visual surface detection) How shadows are cast on the opposite side of an object and a light source and so on.

From simply the amount of time it would take to create our own “engine” or at least standalone program not running from any game engine dynamic linked library, we can easily conclude that we absolutely need a ready-made game engine. But which game engine? Most projects that are undertaken by low funded teams are made with one of two very popular game engines. These are the Unity3D Engine and the Unreal Engine 4 or the Unreal Engine 3. This is due to approximately 95% of their functionality being completely free to use for the public experimenters. There are other game engines available for purchase, the most popular of these being the CryEngine V, which is responsible for all of the stunning and spectacular environments displayed in every Crysis game. The Source Engine 2.0 which is an older and versatile version of the engine that brought us the Half-Life 2 game, Team fortress 2, and Portal. The Leadwerks Engine which is mostly used by beginner game developers as it is an easy to use engine which hasn’t made any extremely notable projects. Lastly Torque3D which is similar to Leadwerks in its mission to make an easy to use baby’s first game engine for college students and also doesn’t really have any notable examples of widespread interactive experiences. Truthfully we could use any of these engines to create our project, but only three of these engines have a native built in understanding of stereoscopic 3D. These engines are the Unity engine, the Unreal Engine and the CryEngine V. The first two are nearly completely free and the third is quite expensive so we can discount it as being viable, however it would be possible to run our software on the Unreal Engine.
The only reason we did not is because of the primary software developer of the team having experience in it. Either Engine would have suited our purposes.

3.9.2 Camera Techniques in 3D simulations

When we say camera techniques, what we mean is the viewport through which a user is able to view a virtual world. It is a requirement to have something like this in a virtual world and every single gaming engine has a base camera class and is even necessary in order to even start building any sort of interactive experience. This fact makes it incredibly important for the camera to be very well behaved.

There is a much more complex take on the control when using it to create a 3-D viewing experience as opposed to being projected onto a monitor or other screen. This is because not only do we need a single camera but we also need a second. Each of these cameras will record and display one unique image per frame, one for the left eye and one for the right eye. The human mind will “mesh” or “combine” these images into one single perceived image and create the illusion of a fully 3-D object.

As discussed earlier, our chosen Gaming Engine is the Unity Engine. The Unity Engine controls the camera based upon a vector to vector travel system. The camera itself isn't limited in speed, direction or frame lag in order to exist where it needs to when it needs to exist there. Usually these cameras are locked at an angle and centered around the player's current position and the player movement controls are nearly directly paralleled to the camera vector class.

The first option is to use the two camera side by side to one another and move them around as a statically paired unit. While this is a legitimate method and does bring the illusion fairly well there are weaknesses in it that betray its nature. The heart of this project was always to do the best possible job to create the illusion of true 3-D and that means no expense should be spared.

This thinking process has led us to the second option. Near-perfect head tracking, and nothing less, would allow for something so immersive no thought of a mistake will break the feel of immersion. Therefore not only will the distance between the user and the screen be tracked but also their vertical height and angle of viewing as well.

3.9.3 Control Scheme

It was a major issue in deciding just how exactly the player was going to interact with the software. Our first thought was computer vision recognition. This would be insanely complicated, and imprecise as was demonstrated by the X-Box Kinect. How this would work is a visible light spectrum camera would take
in the image directly in front of the device, and be programmed to recognize and track a human shape. Also differentiating between the hands, legs, head and torso so as to see when a hand would move over a specific area. Also before any use the camera would have to be calibrated to a specific user's height and distance from the device. When a player would raise their hand, the camera would send the frame by frame data to a powerful processor which would tell the interactive experience that a hand was being raised in a potential input area. This would need to be incredibly precisely programmed and quite honestly it was far beyond the time and man-power the team could muster at the current date.

Another option is a handheld device whose position could be tracked. This is similar to the HTC VIVE controllers and is in fact where we found the idea. Infrared radiation cameras were already a possibility so this could easily be implemented from a hardware standpoint. However without seeing what the input zone is, the control becomes far less intuitive. Not only that, the hardware would need to track these devices just as precisely as the user’s head itself. This would most definitely require even more capable hardware or simply more copies of the same hardware, both of which would push the work time to breaking point and the potential issues to maximum.

There are standard ways of playing an interactive experience. These usually boil down to a keyboard and mouse or a gamepad. A keyboard would provide much more versatility when it comes to input as it is much larger and is a much older form of control than the more recent variations. However the gamepad has a distinct advantage as it can be held in the hands of the player while it is used, unlike the keyboard which must remain stationary.

The unity engine has many built in interfaces for use in its simulations but this time we decided to use what amounted to be an xbox controller, but without the brand X-Box on the controller, we can use a standard gamepad as pictured below.

3.9.4 Necessary Computer Specifications

Deciding exactly what would be the minimum processing power, main memory and graphical technology wasn’t so much a decision of what I wanted it to be as it was a decision of what frame rate and loading times we were prepared to accept if worse comes to worse and the inevitable march of time moves past us as I’m sure it is already creeping up even as you read this paper.

Now just because I said it was easy doesn’t mean it was easy to swallow. We decided in order to make the chance of headaches or motion sickness at as low a minimum as we possibly could, the FPS could never dip below 30 FPS and should aim to stay at around 90 FPS. This is a very interesting issue, as a
computer that can comfortably stay at 90 FPS is rare and expensive. Thankfully every member of our team understood this and two of us had personally built computers that could withstand this kind of high stress and backlog program.

One option was to take the base design of the computer we were already familiar with and model it up from there but this came with challenges of cost. Starting with the most basic starting question of building a desktop computer. Would we use an AMD motherboard and CPU or would we use an Intel motherboard and CPU. This is a monumentally huge difference given the correct circumstances so we'll need to look at each carefully.

The first option was an AMD CPU and by necessity an AMD motherboard capable of running an AMD CPU. The AMD line has for a long while been the less expensive and slightly less powerful of the two major CPU architectures in the desktop PC industry. This is mainly due to AMD using more cores that are less powerful, to outnumber the Intel architecture while still having the capability to run many times more background processes simultaneously. The idea is to lessen the load of each core so that the cores running the main program are freer to focus on the tasks assigned to it and therefore get more frames per second with more complex tessellation, more defined shadows, more hair physics and so on and so forth. While in theory this would be a superior design for the gaming needs of the average desktop PC gamer in practice it isn’t. This is because the way interactive experiences are developed now focus heavily on internet connection multiplayer interaction and competition. This means that synchronization is imperative, down to the milliseconds lost being the difference between a successful project and a failure. Synchronization between two CPU cores even in the same computer is a monumental task on its own for the operating system to handle, let alone two different computers. Thus the less synchronization the computer has to do for its own cores then the less it has to worry about synchronizing the frames it is building based upon information collected from a server or host of a network.

From this we can safely say that an Intel CPU and motherboard will be a more compatible fit with our needs. Which leads to the next choice, what graphics card will we choose? We could use the AMD cards or we could choose the Nvidia cards. The same more core with less power concept is present in the AMD cards precisely as they are present in their CPUs. As a general rule of thumb, AMD is more suited for an AMD board and CPU and Nvidia is suited for a higher performance Intel CPU and motherboard, so our choice is clear.
4. Standards and Design Constraints

4.1 Standards

4.1.1 Indian Hill C Style and Coding Standards

The Indian Hill styling and coding standards are set of guidelines on writing C code for AT&T Bell Laboratories' Indian Hill community. The coding guidelines are a C styling guide focused mainly on UNIX systems, but can be applied more generally to most C code. The standard use provides neat, organized, and readable code. This, in turn, leads to well understood, more stable, and less error-prone code. The Indian Hill Standards are one of the more commonly known styling guides for C.

Beginning, the files themselves are recommended to be no longer than 1500 lines. The standard recommends that for workability within the files. In addition, it recommends avoiding lines longer than 80 characters. The file extensions are then the commonly used ones as follows:

- *.c for C source files
- *.s for assembly source files
- *.h for header files
- *.o for object files

Abiding by a variation in the order of the description and include sections, at the top of each file a description specifies the objects and descriptions therefore. The list of includes then follows. Next, any typedefs are defined. Global data definitions are declared afterwards. Any definitions should be defined. And finally the functions are then specified. This format allows for a consistent, and logically read flow of the file format.

Header files should contain data types, system includes and definitions needed by data types. It is important to note that the Indian Hill standard does not specify use of an include guard. The intent is that nested includes are avoided as much as possible so that typedefs and initialized data definitions are not see by the compiler twice. We see this as sometimes unavoidable and better practice to use include guards, and make the modification to the guide by using guards.

External definitions variables start from column and are each declared on their own line. Comments should be used to describe the use themselves, unless constants are self-documenting.
Comments which describe either algorithms or data structures should begin with the multi-line comment start characters (\texttt{/*}) and the multi-line comment end characters (\texttt{*/}) each on their own line. The comment is then placed on lines in-between, where each line starts with a space an asterisk. Each asterisk lines up on the second column. Short comments can be an exception to this rule, and placed on a single line (still using the start and end multi-line characters, but on the same single line). Very short comments can be placed on the same line as a statement of code on the right. This keeps the formatting of the comments neat and clear.

Every function starts off with a multi-line description which contains the name, the description of the function. The return type should be given its own line followed by a comment of the return value description. If the comment is large, it should instead be placed in the function description. The name and parameter names are then placed on their own line. Following, each parameter is given its own line and described with a side comment. Local variables are placed after the opening brace which is separated by a blank line, along with a description of each variable. Extern variables are also placed on their own line. The standard then follows a tabbing scheme where declarations and nested-statements are tabbed once from the left.

Nested statements should open with a left brace on the same line as where it's used. The closing brace should be on its own line unless used on an ‘else’ or ‘while’ in an if-else or do-while statement. ‘case’s are placed on the same tab-indent level as the switch statement they are declared in. Fall-through should be used sparingly for easier management.

As far as the naming conventions, the Indian Hill stylizes some general rules. No underscores should be used. Typedefs and defines should be in all-caps. Functions, macro functions, variables, and structures should be in lowercase. This is usually common practice in C.

This set of standards also specifies several special case rules. For brevity we will not go over these. But with the existing rules, this then completes the standard. It can be seen how the standard provides readability throughout the system.

\textbf{4.2 Health and Safety Concerns}

\textbf{4.2.1 Motion Sickness}

According to a study done in 2016, a turbulent flight can cause above twenty five percent of passengers to experience some or all symptoms of motions sickness. Clearly, this is something we hope to avoid in our project as it is hard to appreciate a well-crafted technological wonder when you’re busy tossing your cookies into a small paper bag.
These same symptoms at about the same percentage of users were reported in the first few months of the release of the PlayStation Move virtual reality headset. Which is comparable to our project as the basic idea is the same, focusing on a single object that is only projected to be where it needs to be when it needs to be there.

But what exactly is motion sickness? Strangely there is no concrete definition of what motion sickness is, or a strict explanation of why motion sickness is present in some humans. Instead there are a wide range of theories on both subjects. According to Professor John F. Golding of the University of Westminster, London, there are two much more common symptoms associated with this phenomenon than the rest, which are nausea and vomiting. The most widely accepted reason for nausea and vomit is an ancient function of the human body to keep you alive. When the dizziness or perception conflicts arise, the human brain might believe that it has been poisoned or had an adverse reaction to a plant or animal food source. The body then wants to immediately get as much of the toxin out of your body as it possibly can. The fastest way to accomplish this is to empty the contents of your stomach onto the nearest friend or relative so that they are not inside you any longer. An unfortunate issue with this theory is that the vomiting seems not to take effect immediately due to our conscious resistance of the subconscious decision our minds have made for us, so honestly the technique doesn’t work as well as its purpose hoped it might. The others which are less common but still troubling are sweating, increased salivation, warmth, dizziness, drowsiness, headache, loss of appetite, decreased sensitivity to odors. In addition to these symptoms there are a few that might be unique to artificial images which can affect the human eyes with eyestrain, difficulty focusing one’s vision upon a single point and these two can lead to moderate or severe headaches.

Professor Golding explains that what we can agree is the most common cause of these symptoms is a sensory conflict. A sensory conflict is what happens when two of your sensory organs report information that contradicts each other. For example your eyes could be telling your brain that you are flying high above the clouds, however your skin tells your brain that there is no wind billowing around you, or your nose smell something other than high altitude weather patterns.

In particular one sense stands out as one the body absolutely hates finding conflicting information with. This organ is your vestibular, or the inner ear canals which tell your brain what was gravity is pulling you in order to keep your balance while walking. The two senses that most often conflict with the first are your ocular senses and your kinesthetic senses. As an aside, your kinesthetic sense is your movement and bodily awareness, which allows you to close your
eyes and lift your arm, and know exactly where your arm and hand is in relation to the rest of your body.

While we try to make virtual reality as realistic and responsive as possible, there are some failings that we cannot consciously pick up on, unless you count ghosting, but our subconscious is fully aware something is amiss. If you were in a virtual world and you moved your head about 1 centimeter to the right, that action would happen immediately to your body, but the response time of the computer program might be 5 to 10 milliseconds after your head adjustment has already happened. Therefore the quickest response to possible is needed to ensure the highest chance of a comfortable and pleasant experience which would not require a small paper bag.

The same effect can be brought about from low framerate, and it can happen very quickly. I have personally experienced this on the numerous occasions the program failed to load frames and when I turned my head to look right the image did not change, both wrenching me quite quickly from the immersion and making me physically stumble and have to regain my standing balance. Therefore a constant stream of frames at a minimum what our eyes can differentiate is a necessity, and this number is 60 frames per second, per eye, which the projector is up to par of doing, now all it needs is a stream of data uninterrupted which is why simplicity will be our best friend in creating this. We do not need fancy textures or scripted set pieces that run on too long. And we do not need to put the burden of constant frame by frame measurement and calculation of “angle of attack” to be put onto the already loaded down processor. This is why we will be implementing a separate board to do a few key functions that will not take up much processing power as much as they will take up time every frame that is produced away from the current desktop processor could be spending generating the camera’s environment for the next frame.

One of the benefits in our design, however, is the ability to view the rest of the real world. As opposed to VR headsets which constrains the user’s ability to view the real world in their remaining vision not used by the display. This reduces the risk of motion sickness by providing the user parts of the real world to balance their internal biological senses.

4.2.2 Eye Strain

As for the eye strain, we have cleverly (or perhaps happily accidentally) avoided this issue. Eye strain happens when the eye must focus on something far too close to itself and compensates by curving inward. This also creates an issue if the object inside the VR world is supposed to be very far away the eye might try to focus on it and get very confused when the world gets blurry for a moment before reverting back to what it was doing. In our case the hologram is actually being projected on a screen from the projector at a minimum of 2 feet back and
the screen to the viewing aperture will be approximately another 1 to 2 feet away from the screen on which we will project the images.

4.2.3 Photosensitive Epilepsy (PSE)

Photosensitive Epilepsy is a form of epilepsy in which seizures are triggered by visual stimuli that form patterns in time or space, such as flashing lights, bold, regular patterns, or regular moving patterns.

The exact stimulus that triggers the seizures varies from one patient to another, but we should be careful with the choice of media we choose to display to people because it could cause trigger people using our platform to have seizures.

The visual trigger for a seizure is generally cyclic, forming a regular pattern in time or space. Flashing lights or rapidly changing or alternating images (as in clubs, around emergency vehicles, in action movies or television programs, etc.) are examples of patterns in time that can trigger seizures, and these are the most common triggers. Static spatial patterns such as stripes and squares may trigger seizures as well, even if they do not move. In some cases, the trigger must be both spatially and temporally cyclic, such as a certain moving pattern of bars.

The first case of epileptiform seizures related to a video game was reported in 1981. Since then, many cases of seizures triggered by video games were reported, not only in photosensitive, but also in nonphotosensitive children and adolescents with epilepsy. Specific preventive measures concerning the physical characteristics of images included in commercially available video games (flash rate, choice of colours, patterns, and contrast) can lead in the future to a clear decrease of this problem. Risks can be reduced through measures such as keeping a safe distance away from the screen (at least 2 meters). This is especially important to us because our design will have to make sure our screen can be viewed from a sufficiently far distance away as to reduce the risk of seizures.

Ensuring a high frame rate is also important. It was observed in that faulty fluorescent lighting was capable of producing seizures. Fluorescent lighting has a flicker rate sufficiently high (twice the mains frequency, typically 100 Hz or 120 Hz) to reduce the occurrence of problems. However, a faulty fluorescent lamp can flicker at a much lower rate and trigger seizures. It is important our display setup does not malfunction and producer flickering light at a low frequency.
5. Design

In the following section we will be going over the design we are considering and the parts necessary to make this design a reality.

5.1 Overview

For our project, we will design a holographic system consisting of a virtual space embedded inside a display shape. The display shape will be a 3-dimensional surface (ex: cube, tetrahedron, etc.) and will allow users to interact with its virtual spaces, each featuring different applications. Multiple users will then be able to interact with a section of the display shape and view a synchronized application for collaboration.

The full 3-dimensional effect can be achieved through head-tracking and stereoscopic imaging. Stereoscopic imaging makes use of a user’s two eyes. One image is rendered and displayed on the left eye, and the other on the right eye. Since the two images are rendered from two different perspectives this creates one illusion of 3-dimensions to the user. This effect is the same effect used for 3D movies at current movie theaters. A lesser used second methods completes the effect. It requires knowing the user’s eye positions and tracks their head movement. By moving around a virtual object on a display, a user can see multiple perspectives of an object depending upon their position.

To achieve the first of these effects a 3D display surface will be acquired (ex: a 3D projector) along with 3D glasses. To achieve the second effect, a compact head-tracking system will be designed to attach to the existing glasses and track head movements (eye position).

The head tracking system can be designed using a form of triangulation. Knowing the 1-dimensional distance between one target point, and at least three different target points, and the relative 3-dimensional positions from each of the reference points from each other, we can calculate a system of equations to retrieve the target point’s 3 dimensional space. This effect can be used to calculate two target points on the corners of the glasses and allows us to calculate a majority of the orientation of the glasses. That is the x, y, z coordinates, yaw and roll. What is missing is the pitch which can be calculated through an additional sensors (the accelerometer + gyroscope).

From here the sensor data will be emitted from the glasses to the rendering computer via a data transceiver. The rendering system will compose of a high-end desktop capable of enough performance to render all graphics, and thus every aspect of the display shape. An engine running on the rendering
system will take the user’s position and render the correct perspective as if the virtual environment was inside the shape. Lastly, rendered graphics are sent out and displayed on the display surface.

Using the rendering engine (ex: Unity), several applications will be designed featuring the key research points of the platform. One application will focus on a design topic. Another will focus on an entertainment topic. And one will focus on interactions between multiple users.

5.1.1 Glasses Detailed Overview

The glasses system is processed through an MSP430 processor. DLP-Link is used to generate the stereoscopic projector effect. This is common for most home-project systems. A sequencing of white flashes indicate to the glasses commands which issue a left eye or right eye projection image. Timings are handling the MSP430 itself which then controls opens and shuts the appropriate LCD shutter. This creates the overall stereoscopic effect. For the IR tracking system, a series of infrared LEDs are placed in specific locations on the glasses themselves. This is then used by the IR tracking cameras on the tracking system. In addition to tracking points, and accelerometer and gyroscope are
onboard the glasses to more accurate calculate angular velocity and acceleration. Using a Kalman filter (mostly on the tracking system, and not the glasses), data is then fused to produce accurate orientation. To take best advantage of the gyroscope and accelerometer, a “pre-Kalman” filter is used. That is, a filter which estimates the covariance and overall change in state. The second stage, main kalman-filter will be used to provide absolute position. This information is then encapsulated via a Data Link Layer, and then transmitted by an IR transmitter.

5.1.2 Tracking System Detailed Overview

Figure 8: Tracking system overview diagram

Figure 3.4 represents an overview of the tracking system. From the glasses, accelerometer and gyroscopic data and received from the glasses. This data is encapsulated in the Data Link Layer, and then fed into the Kalman filter. In addition, IR trackers are seen from an IR Camera. Vision processing is applied through a blobbing algorithm on the Raspberry Pi’s GPU which generates the UV coordinates of each point. To convert from 2D space to homogenous 3D space, the Camera’s orientation is calculated during a calibration process. This generates the Homogenous tracking position. From here a system of equations can be used to estimate the tracking position. If multiple cameras are present, data needs to be sent from one camera device to a central camera device. This
occurs via a set of SPI serial wires. Messages are transmitted from the rendering system to set calibration settings, and begin any calibration modes. This information is also relayed to non-central tracking devices. Once the tracking data and IMU data are processed, a Kalman filter is applied for better estimation of orientation. Finally, a predictor is used to calculate the future state of the system to compensate for any latency in the system overall. This is then given to the rendering system via USB OTG.

5.1.3 Rendering System Detailed Overview

![Rendering System Diagram](image)

**Figure 9:** Rendering system overview diagram

The rendering system is in charge of using the head-tracking data to render the correct projection of the application and display it to the user to create the full 3D effect. The system first consists of pre-processing the received head-tracking orientation from the tracking system. This information is received from USB OTG by a COM-Access DLL library. Unity (the main application), requests this information and runs a projection algorithm to set up a set of cameras making up the stereoscopic effect. The rendered projections are then pushed to the Projections by HDMI. In addition to the main applications running and the rendering of the projections, audio data is passed to speakers. To control the application, a USB controller is given to the user and input data is processed by the application. Finally, to control any calibrations and configurations of the head-tracking system, a configuration menu is placed in the application such
that the user, or administrator of the system can run the calibration process. The control between the application and the tracking system is handled through the COM-access library.

5.2 3-D Effect and Head Tracking Concept

The Full 3D effect can be achieved by projecting the right image on the projector for each eye. Abstracting away the 3D active shutter system, we can think of the projection system as two images being projected onto the display; each eye of the perceiver only seeing one of those images. With stereoscopic 3D, perceived depth is added to the image. This is conceptually rendered by replacing a single camera in our 3D application, with two cameras moved horizontally away from each other and offset similar to how the human eyes are offset. Thus, each image is rendered as if the perceiver was at the position of the camera inside the application. This creates the full illusion of depth in the virtual environment.

In real life, when a user move their head, the object they are viewing changes in shape. A perfect example of this is looking at a cube. If looking directly at the top of the cube, the overall cube will look like a square. But from different angles, the top of the cube will be skewed, and the other sides will be exposed. If an object were physically inside the display, it would behave like this, not a static perspective image. To correct this, we can track the user’s eyes as if they were cameras in the 3D system such that when the user moves, the cameras move in the same motion as the user’s eyes. Since the user’s eyes are static in reference to the head, only the head’s position is needed. (A note on different eye positions is described in limitations). This overall process is called head tracking.

Up until this point we were only focused on what was rendered onto the display. Everything would look accurate if the display was always in front of the user as it is with VR headsets. But since the display is in the user’s 3D space, as the user’s eyes move the perspective of the display changes. For instance, if the display was a flat triangle, the triangle would contract if the user moved away from the normal of the triangle, and expand when moving closer to the normal. In addition, if the user moved further away from the display it would appear smaller in their field, than if it did closer.

To conceptualize the effect, what’s happening is that the display acts as a window from the real world into the virtual world. From an application’s standpoint, the important part is where that window is placed to view everything inside the application. From the display’s standpoint, processing is needed to determine how to project it onto the screen as if it were a virtual window.
5.2.1 Projection Algorithm

To describe the algorithm we will first simplify it by abstracting away the stereoscopy. This is easily achieved by performing the same algorithm twice; one for each camera and eye pair. So now we can only focus on a single perspective.

Now before we move on how to manipulate the system, we need to understand the 3D rendering system. Objects in the virtual environment are made of a series of points in 3D. For an example a cube with 8 points on the corners. These points are then transformed from a central origin point in the 3D object model, to a position and orientation in the 3D world (Ex: placed 10 unit above the ground and rotated 10 degrees around the y axis). When rendering, these points are the converted from the world coordinate system to a coordinate system in reference to a camera. The world-to-camera matrix (sometimes called the view matrix) does the transformation from the world coordinates to the camera orientation. Finally, a camera projection matrix is used to transform the 3D coordinates onto the 2D camera rendered image. A good model of this is viewing region created from the camera in between planes; a plane close to the camera which starts the viewing region (near plane), and larger plane further back which ends the viewing region (far plane). This is commonly known as the viewing frustum, since the two planes create a frustum (the shape of a cross-section of a pyramid).

![Figure 10: The viewing frustum](image)

As the model coordinate do not change, we will focus on the view and the projection matrices. The world coordinates will be standard Cartesian coordinates with some unit, and an origin at the center of the base of the...
display structure. This is for simplicity and multiple displays can reference a single central point. The camera orientation will be statically positioned at the center of the display surface. The intent is to render from the window to the virtual environment. What’s left is to setup the projection matrix.

Going back to the frustum model. The near plane is trivial and can be setup by using the display surface boundaries themselves. That is, the rectangle of the display is the near plane. This however means we cannot render anything appearing outside of the of the display, but is also convenient since we do not have to worry about anything extending outside the display from physically clipping off the boundaries of the display region (we will discuss more on this in Limitations). The far plane can be calculated by introducing a maximum distance constant \( z_{far} \). This represents the distance between the centers of the near plane and the far plane. Typically, this will be very large so that far objects can still be rendered and displayed. Due to the projection onto the display when viewed from an angle, although the far and near planes are still parallel, the far plane will be offset along itself to create the head tracking effect. In particular, the far plane’s position can be calculated by knowing the position of the perceiver and calculating the intersection of the lines draw from the perceiver and the four corners of the near plane. This produces the overall effect when the the projection matrix transforms the camera’s perspective to match the viewer’s perspective when projected onto the display.

![Figure 11: Change in projection from user’s movement](image)

64
5.2.2 Head Tracking Algorithm

\[ \hat{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \]

Let’s begin by defining tracking points. Call it \( \hat{p} \) where \( \hat{p}_i \) is a tracking point in 3D space with coordinates \( x, y \) and \( z \). Now each point is tracked by a different sub-algorithm (described later) and seen by camera in a 2D coordinate system. We will call these points \( \begin{bmatrix} p_u \\ p_v \end{bmatrix} \).

Before we go further, we have to come to the realization of a few things. Everyday cameras are imperfect. They have a variety of distortion to them, so it’s not as simple as directly using an inverse camera matrix for determining the point positions in 3D given a 2D projection. Instead, we need to calibrate the cameras to determine distortion coefficients as well as the camera matrix. We took OpenCV’s approach to correct camera distortion as it is widely used, researched, and has a test pattern for easily calculating the distortion coefficients and calculating the camera matrix. When a particular test pattern is held up to the camera, the coefficients can be calculated and averaged out across several test frames for an accurate representation of the camera.
The distortions themselves are manifested into two different types: radial and tangential distortion. Radial distortions are caused by the lens used for the camera's focus. They contribute to a “barrelling” effect where the image is curved and distorted into a barrel shape. They can be corrected by using the following formula:

\[
\begin{bmatrix}
  x_{\text{corrected}} \\
  y_{\text{corrected}}
\end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

\[
r = \sqrt{x^2 + y^2}
\]

Radial distortion correction equation

Tangential distortions are caused by the imperfection in placement of the lens. When placed, the lens is not always perfectly parallel to the lense, and needs to be correct. Tangential distortions can be corrected using the following formula:

\[
\begin{bmatrix}
  x_{\text{corrected}} \\
  y_{\text{corrected}}
\end{bmatrix} = \begin{bmatrix}
  x + 2p_1 xy + p_2 (r^2 + 2x^2) \\
  y + 2p_2 xy + p_1 (r^2 + 2y^2)
\end{bmatrix}
\]

\[
r = \sqrt{x^2 + y^2}
\]

Tangential distortion correction
Now most distortions are corrected and we can continuing tracking.

Next, we need to adjust the camera’s coordinate system into the coordinate system relative to the base. This is calculated by multiplying by a camera rotation matrix and adding a coordinate offset, similar to the world-to-camera transformation matrix described in the projection algorithm. The given offset and rotation can either be physically measured, or more accurately calculated in a calibration process (described below).

At this point we need to transform the 2D coordinates into 3D space. From the camera we create a line in 3d space extending out and away from the 2D Point. That is to say:

\[
\lambda \begin{bmatrix} p_u \\ p_v \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}
\]

Where \( p_u \) and \( p_v \) are the image coordinates as before. This leaves us with the linear equation mentioned above, but without a precise point; only a range of points. However, if we consider multiple points and know the positions relative from each other point and the orientation, we can infer the depth.

We can represent the orientation of the viewer’s eyes as a distance relative from a central point on the glasses. These points will be adjustable to match the viewer’s pupillary distance (distance between the eyes), distance between the eyes and the glasses (perpendicular to the eyeglasses plane), and the distance
vertically from the glasses's central point to the viewer's eyes. This means we only have to focus on tracking the central point on the glasses. The position of the glasses can then be represented as cartesian coordinates and the rotation as a normal to the glasses frontal plane and a rotation around the axis. The IR tracking, accelerometer and gyroscope will assist in approximating the glasses's coordinates and directional vector.

5.2.2.1 Kalman Filter

Taking in all this information into account, we need an algorithm to “merge” all of this information. We take inspiration from a Kalman filter. Kalman filters are commonly used in areas where multiple non-perfect sensors exists and contribute to redundant information, such as robotics and GPS systems. Kalman filters predict next positions, then use an interpolation of existing sensors, and the prediction to compute a more accurate state. The GPS example being, using an accelerometer to predict the next coordinate. While more accurate than a GPS, accelerometers tend to drift over time. Over time the accelerometer drifts, and the GPS then becomes more accurate. The Kalman filter takes advantage of both cases and statistically analyzes the best results, and converges on the best interpolation between the two.

For our purposes, we have much more to take into account. Throughout our system we have a lot of unknown variables. Cases such as the trackers where we can only infer the depth (as opposed directly measuring it. The entire system can be easily solved using a system of equations. We could even add in a least-squares algorithm for further accuracy. However, for more accurate results we could put them into an adapted Kalman filter and make better use of all our sensors. In addition, we can even predict future movement of the user which is important given the latency of the system (discussed further in Limitations).

Kalman filters are designed to be used in linear equations. Unfortunately, many applications, including our case, are non-linear. The more non-linearity in the use and pre-processing of the sensors leads less optimal results. Variants of the kalman filter have been used to correct for this errors. A common implementation known as the extended Kalman filter is widely used among this. The extended Kalman filter makes use of the Jacobian to linearize some equations used in the Kalman filter. We can there apply this to our nonlinear usage.

Another modification we need to make the Kalman filter is the use of time dependent estimates. As data comes as the camera sensors, sometimes we need to poll the head tracking state faster than what the sensors can produce. Thus, leaving the camera tracking data to become less relevant to the
acceleration and gyroscopic data that comes through. In a multiple-camera system this can be worse since the two cameras can be out of sync with each other; appearing at different times. Finally, we also want the ability to make predictions of the tracking state ahead of time to account for latency. The latter is previously solved through a simple method, which is to use the state-update over the course of a set time to make the prediction, such as using the previous derivatives (accelerations) to predict the next state. The former has a more complicated solution which involves manipulating the Kalman gain.

The Kalman gain is a series of correspondences representing the interpolation values between all sensor data and predictions. We wish to manipulate that value in steps, and put less gain on the sensors that have not been updated in awhile. Once a sensor is remeasured we can rest the gain on that correspondent and repeat the process. We can model the decline of the gain as a linear, or better, an exponential decay function or a reciprocal function. This kalman gain can then be normally updated on every full step the sensor data is received. While every in-between step the decay function is better estimated.

Finally, we have sensor latency. Different sensors in the system have different latency values depending on time required to process. An easy way to correct sensor latency is to delay all sensor data until all data is instantaneous to a single moment. This however, throws away temporality of the data that could be used for better estimates. Ignoring the difference in sensors is the best solution, not only is it the easiest, but it allows the Kalman gain to set the temporal significance well.

5.2.3 IR Point Tracking Algorithm

The algorithm for tracking points is simplistic on a high level standpoint, but we take advantage of special hardware to optimize it for speed.

From our hardware constraints, we chose to use an infrared system for tracking. This means we needed a system which detected infrared tracking points. Naturally, we went with a Raspberry Pi (RPI) Zero with a NoIR camera. The RPi Zero gave us the flexibility of various I/O, including a USB serial port we could directly use to talk to the main rendering system computer, all with a relatively low cost. In addition, we could also use the RPi to offload some of the calculations. This is also important due to the high data throughput requirement of camera data we would need to transfer to the main rendering system. From here, the NoIR would pick up the majority of the Infrared track points, but as well as visible light since the NoIR is a bit of a misnomer in the fact that it lacks an IR filter. That is, IR light is not filtered out from the camera, but visible light is still included in the NoIR's spectrum. To accommodate for this, we added an additional filter which filters out the remaining visible and infrared light that differs from the track-points frequency, leaving only

69
high-intensity visible light, and the frequency used by the track-point light. This creates the assumption that the cameras will only receive light from the track-points, and some minor background noise.

Now we provide a high-level overview of the point-tracking algorithm. First, we wish to convert the 3-channel RGB image into a binary image representing the areas of interest with enough intensity to be the track-points. Ideally, this will filter out most background noise. However, it is still important to note that high-intensity light sources may still appear. For simplicity, we assume that there are none of these sources our in the field of view for the camera. This is still a proof-of-concept design, and can always be improved. Once a binary image is created, we can collect all the above-threshold point locations and collect them into a smaller array. This will reduce the processing time than continuing processing on the large image. Once collected, points can then be grouped based on distance to form “blob”. Each blob can then be assigned an id, a center point based on the average point location for sub-pixel accuracy, and a size. A size threshold can be put into place to further filter out background noise. Then, each blob can be assigned a physical tracking point based on relative orientation, the previous location of each blob, and velocity/acceleration data.

Before we continue, a brief overview of the RPi’s computational hardware is needed. The RPi runs on a Broadcom microprocessor which is composed of a 1.0 GHz ARM microprocessor and a GPU specific to Broadcom known as VideoCore IV. The VideoCore IV contains various hardware to process not only graphics, but video codecs and camera interfaces. The section of VideoCore IV we will focus on are known as QPUs (Quad Processor). A QPU is a 16-way Single-Instruction, Multiple-Data (SIMD) processor. Each QPU is comprised of 4 SIMD processors whose instructions are multiplexed over 4 clock cycles. Therefore, each instruction conceptually takes 4 clock cycles and is executed across 16 operands. Groups of four QPUs have additional hardware for special memory access and additional operations, and in total, are known as slices. The RPi contains 3 slices for a total of 12 QPUs. There are two ALUs for every QPU: an “add” ALU and a “mul” ALU. With this, for every ALU QPU instruction, multiple ALU operations can occur; one operation from the “add” ALU and one from the “mul” ALU. Since the clock rate of the RPi Zero is 400 MHz, this gives us a total potential of 38.4 GFLOPS.

As implied above, the RPi’s GPU has significant speed advantages when compared to running on the CPU for large data operations. As such, we can use the GPU for the cost of a small latency, and difficulty. Given the large amount of pixels needed to be processed (62.1 MP/s), the latency cost is small when compared to running the algorithm on the CPU. Given that this project is also an academic exercise, it is useful to test low-level and parallel architecture skills. Finally, the latency of the entire system needs to be minimized to make the
most accurate display rendering (and prediction) so that the head tracking effect is of high quality, and user can experience a smooth response.

The algorithm first consists of retrieving raw camera data in an RGB format. This is discussed later in Camera. Once the data has been put in RGB format, it can then be directly operated upon.

First we need to convert each data piece into binary format. We do this by applying a thresholding algorithm which converts the image to grayscale and checks that the image is above a specific threshold. We have the option to do one of several things. The first is taking the brightest of a single channel. This offers the quickest solution as we directly reduce it to a single component without extra calculations. The second solution is to perform an average of the three channels by converting the image to grayscale. This offers a more dynamic approach by using all three channels and offers better resolution. Next we could also perform a weighted average using the RGB components of IR. The RGB components of the IR used (940nm) can be seen the figure below. This offers a better solution when other colors are shown besides the IR used in the system. In addition, it also gives a greater dynamic range. Finally, we could perform a weighted average, but linearly penalize colors that do not match the expected RGB components of IR (Ex: removing all only red pixels). This offers a better thresholding when other non-IR colors are shown. Due to the fact that we desire speed the most, high-resolution thresholding is not needed, and the non-IR background noise is minimal, we decided to simply take the blue channel.

![IR RGB Components](image)
Figure 14: Infrared RGB components as detected by the IR camera system (940nm)

Figure 15: Infrared LED and colors at various luminosities as detected by the IR camera system (940nm)

Now that we have the binary inputs, we can condense the existing array. We can perform this action in one of two ways. The first is to use a prefix sum algorithm. That is, iterate through the array and give each positive pixel an index, and increment the index. There is an $O(\log(n))$ parallel algorithm for this which makes good use of the QPU parallelism. However, once the indices are computed they need to be written back to memory. This may introduce larger delay than the next idea. The next idea is to simply continue the program on QPUs where at least one of the 16 pixels being processed in a QPU is available. Although extra memory writes are not needed in this method, excess calculations are introduced for pixels which have negatives in their 16-pixel QPU pipeline. Although these calculations are redundant, it still may prove faster than memory depending upon lookup time. With both methods, it is important to place a maximum limit as it may drastically slow down the processor when a large portion of the image is positively detected.

Figure 16: Parallel Circuit representation of a 16-input prefix sum
Once properly indexed, the pixel coordinates can be stored and used to group pixels together. This can either be done on the host CPU or the GPU device. The GPU version will first assign every positive pixel a unique index (the index of the condensed array, or the full array depending on the last choice). With this, pixels can be grouped together when two pixels are within a specified threshold away from each other. Iterating through several passes of this algorithm, each group checks to see if there are any other nearby pixels and adds nearby groups to itself and increases its radius threshold proportionally. The CPU host version is very similar. The algorithm tries to group pixels that are near each other into blobs using a near threshold. Groups are then created, and more and more pixels are added to the group. What remains is a small number of groups that can be tracked.

Finally, the groups’ average center points are calculated. The camera distortion correction is applied just before calculation the average. Using the currently estimated tracking points, a reverse calculation can first be done which retrieves the expected u-v coordinates of the current position. The group's center points attempt to match with these expected points based on nearest distance up to some threshold. If no match is found outside of a threshold, loss-of-tracking occurs. Upon initiation or lost-of-tracking, a different matching method is used. This method is based on matching assuming a standard orientation of the glasses and trackers thereof. The standard orientation is the orientation of glasses when laid on a flat surface. (i.e. the head is upright). Normal tracking then resumes.

5.2.3.1 Camera Details

The camera data can be interface from using a modified version of an existing program on the RPi known as raspivid. Raspivid is open source and contains some examples of how to get raw data from the camera. That being said, it's still not an easy task. Much information is still vaguely documented and very low level. Raspivid makes use of the MMAL API (Multi-Media Abstraction Layer). It is an API specific to Broadcom microprocessors which attempts to simplify low-level functionality of media (encoders, cameras, etc.). MMAL runs on top of OpenMAX (Open Media Acceleration), a lower level interface to media. Now modifying raspivid, we can create callbacks for when camera data frames are received and processes them one-by-one. We can even leave them in the GPU so we do not have to redundantly keep them in the host. Using MMAL and modifying Raspivid we can successfully access camera data.

5.2.4 Data Serialization and Interoperability

Throughout our system, data exchanges occur periodically throughout each subsystem. Our accelerometer and gyroscope will send data to the camera system for tracking and further processing. The tracking system will then
exchange data within itself to the devices which make up the sub-system. Finally the fully-processed head-tracking data is then sent to the rendering system. To successfully communicate, a well defined communication protocol needs to be established for successful and reliable data transmission.

5.2.4.1 Start/Stop

As we send data, we first need to successfully quantize information and frame data so that it can be separated from a data stream. A common way to define frames is by adding a start frame delimiter character to the start of the packet. This indicates the start of a new series of data. However, to distinguish a start frame delimiter from a character of data inside the frame, all occurrences of the start frame delimiter character need to be removed from the data it contains. The Consistent Overhead Byte Stuffing algorithm allows us to perform this reliably, and efficiently as it only appends a single byte. The COBS algorithm itself is applied by replacing the occurrences of the delimiter character with the relative location of the next would-be occurrence. Finally, the location of the first occurrence of the frame delimiter is appended to the start of the frame. From here the actual start delimiter is appended to the beginning. The decoding process begins when the delimiter character is at first seen. COBS is then decoded by retrieving the next relative offset to an occurrence of the frame delimiter, with a frame delimiter which now represents a data byte whose value was the frame delimiter. With this, we can successfully define a quantity of data.

5.2.4.2 Error/Check

Throughout our system, we may sometimes run into errors when transmitting physically. To avoid this we can apply error checking algorithms which can verify with a very high probability that the data sent was either valid or corrupted.

Given the small amount of data we are transmitting, a 2-byte, 16-bit CRC will suffice. The CRC or Cyclic Redundancy Check is algorithm designed to detect probable errors in the validity of a data piece. After specifying a polynomial, it can then be calculated given an array of existing data to create a 16-bit number. The validity status can be checked any time by re-evaluating the calculated CRC and comparing it to the appended, valid version. If data is ever corrupted, the CRC is designed to fail given a small amount of bit change errors.

Implementing the CRC gives us a way to validate data. If it fails, data could then retried, or ignored, depending upon importance of the data.

5.2.4.3 Data Types and Structures

Throughout its use, the system uses various specific message types for interoperability. A trivial way of message structure is hard coding a single
structure between two endpoints of the system. This works until we wish to send more that one type without reinitializing the communication. In most cases the system will not need to specify which type of data is being sent, as the structure is assumed. But only one message type can be used. To add multiple messages types, we simply add another data byte to the beginning of the frame's containing data. We can the parse the type when a message is received.

The infrared transmitter transmits gyroscopic and accelerometer data from the DLP glasses, to the tracking and sensor processing system. This is performed over air, so the stream framing using the COBS algorithm applies well. The data message structure is therefore made purely out of 16-bit data for each of the 6 measured axes in the combined gyroscope/accelerometer, listed in the order ax, ay, az, gx, gy, gz. All communication within the system will be defined as big-endian for consistency across systems. This means the total byte size of the accelerometer data is:

\[1 \text{ Delimiter} + 1 \text{ COBS} + 2 \times 6 \text{ Accel}_\text{Gyro}_\text{Data} + 2 \text{ CRC} = 16 \text{ bytes}\]

After transmitting at 60 hz, the final data rate needed is:

\[16 \text{ bytes} \times 8 \text{ bit} / \text{ byte} \times 60 \text{ hz} = 7.68 \text{ kb/s}\]

5.2.5 Overall Calibration Process

The calibration process allows us to setup the system for the most accurate readings. It involves a bit of manual procedures, but for the most part, only needs to be setup once. However, some per-user calibrations are still needed, and are thus, needed only when a user is switched. The procedure involves several configurations which are statically used throughout the system during processing. These variables include the camera distortion calibration, the camera orientation, the IR tracking constants, and user head/eye metrics.

The first step is to calibrate the camera distortion. This is unlikely to change unless the cameras are physically touched. However, a good example might be when the IR filters are added. Before the calibration can occur, the IR filter must be removed on the camera to allow visible light. OpenCV can then be used to calculate the constants over a series of frames and average the constants to achieve even more accurate results. From here, the camera distortion constants are fully measured. Ideally, commands can be sent from the main computer to each raspberry pi to better automate this process and save time.

Calculating the camera orientation is the most difficult part. Under an ideal scenario this is calibrated once. A production version might be designed so that the cameras are permanently fixed in place, but under our proof-of-concept design, we will use a physical tool to assist in the calibration process for the
cameras. The tool will be constructed with exact physical dimensions such that we can line it up with the base of the display structure, so it is known exactly where it is in 3D space. At the top of the calibration tool will be a test pattern (similar, if not the one used for calibrating the distortion coefficients). We therefore know where every interest point on the test pattern is physically in the real world and can use this to infer the camera's orientation. That is, the camera's 3-variable position and 4-variable directional quaternion.

The next step is to set the IR tracking constraints. The tracking constraints are constants such as the light intensity threshold and pixel grouping distance threshold. These constants should remain the same throughout the course of the projection, but adjusting to different lighting conditions in different environments may be necessary if background noise in the specific infrared band is high. Therefore, it is also convenient to be able to set these from the main computer, but is not necessary.

The remaining configuration is left up to the user. Before interacting with the device, the user must configure their assumed eye distance. Several calibrations can be done to ask the user to see how far away some object is, or outline the display shape with a virtual boundary from various view points. In a production setting, users could create profiles and save them for the next time they use them. Leaving most things a one-time setup, whenever in a production environment or on first use.

5.3 Hardware Design

From a hardware perspective there are a few requirements that need to be met for this to be a successful project. These are the main areas where the hardware will be most intensive and necessary for success.
5.3.1 Final Schematic

As a whole the complete schematic is large and can be a bit confusing, however the final schematic can be split into 3 main portions: the power management system, the IR LED and DLP Link Communication, and the liquid crystal lense functionality. Each of these portions are technically managed by the TPS65835 main hub, but each required their own specific passive components and circuit architecture to function correctly making them basically independent systems.

The Power management system begins in the upper left portion of the schematic and it includes the 5V main power source through the use of a micro-usb connector, the battery and voltage controllers of the PMU, a thermistor for measuring battery temperature, and the actual battery itself. In this section we used a lot of pull down capacitors to level out voltage going through our power pieces, included the use of a thermistor to check battery temps and adjust the charging rate based on the corresponding temperature, and included a green LED for showing power is on. This LED will be able to change state based on the mode the glasses are in. Blinking for charging and sold for normal use.
The IR LED and DLP-Link communication can be found entirely on the right side of the schematic under parts XR1 and T1 and T2. XR1 is our chosen DLP-Link receiver and includes an internal filter for demodulating the incoming signal and filtering out any noise. Following our filter directly we included another low pass filter for extra filtering and making sure the signal was able to be read to the MCU. On the IR communication side, we decided to utilize separate batteries to power the LEDs and save main system battery life. These two batteries would be replaceable CR2032 for long life and little need for replacement. These LED systems are switched on and off through logic level MOSFETS controlled through the MCU. This gives us the freedom of coding our IR Pulses for data transmission.

Our final system is the Lense functionality. Although majority of this is internal to the TPS65835, our lenses are controlled by the internal MSP430 and shuttered through a full H-bridge switching configuration. The results of this switching are outputted through the boost converter out and the LCRP, LCRN, LCLP, and LCRN pins. All you need to do is set the timing in the code for the MSP430 and the system handles the shuttering at the specified frequency.
5.3.2 Printed Circuit Boards

We will be developing our PCBs and having specific components placed on the board for us. The main parts that we are going to chose to be assembled for us are the large surface mount components like the PMU/MCU IC and any other hidden pad ICs that we may have.

As far as what company we may be ordering from, we have a couple in mind. One of the big manufacturers that produces boards in the U.S and offers reasonable prices for students is Advanced Circuits. Because they are based in the U.S they also offer rush and faster turnaround time just in case we need another board soon or quickly. Based on past experiences with them they provide good boards.

Outside of the U.S, Chinese producers offer some of the best prices on PCBs at the price of time. Ordering a board from China will take much longer to arrive when compared to ordering from U.S manufacturers. The quality of the boards is normally pretty good however some people have had issues with quality before.

One last quick method of getting a PCB is printing and etching your own board. This provides the quickest turnaround time for easy testing and at the lowest price. Of course the main caveat is that normally the quality of the board will not be of the same quality as a professionally cut PCB.

In the end we decided on ordering from the U.S manufacturer OSH Park. Because our boards were so small, they provided us with the best option quality and cost wise. At 40x30 mm the total cost of our boards came out to only nine dollars for three and were of great quality and an interesting purple color.
Below you can see a picture of our board layout on eagle and the final board we received from OSH Park.

**Figure 19:** PCB Layout and Final Board
5.3.3 Hardware Parts List

<table>
<thead>
<tr>
<th>Part</th>
<th>Function</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPS65835</td>
<td>MCU/PMU</td>
<td>1</td>
</tr>
<tr>
<td>LR2025</td>
<td>Battery</td>
<td>1</td>
</tr>
<tr>
<td>940 nm IR LED</td>
<td>IR Transmitter</td>
<td>2-3</td>
</tr>
<tr>
<td>Existing LCL</td>
<td>Lenses</td>
<td>2</td>
</tr>
<tr>
<td>Existing Photodiode</td>
<td>DLP-Link Sensor</td>
<td>1</td>
</tr>
<tr>
<td>SMD RGD LED</td>
<td>Power LED</td>
<td>1</td>
</tr>
<tr>
<td>Various resistors/capacitors</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 9: Hardware Parts List**

The above table highlights the main components we have chosen for our 3D-Glasses design. By keeping our parts list relatively compact we are able to ensure a smaller surface area taken up by our PCB thus reducing the total package size in the end. We have also elected some of the most compact yet fully capable parts we could find to maximize the amount of space and functionality we have in our 3D glasses.

5.4 Software Requirements

Even the best hardware is worthless if not utilized by proper software. In this section we will explore the requirements of software needed to make this project work.

5.4.1 Universal Game Idea

This is not in any way an engineering problem but it is a problem, or perhaps just a choice, that must be made. If this project were to be marketed to a wide variety of possible consumers or contractors, it needs to be impressive, and it needs to be understood quickly and draw a user in. Programming this demonstration subject to both impress and draw in falls to me, Matthew Hosken, so let’s go through our possibles and come to a conclusion.

The first thing we thought of was chess. A chess interactive experience seemed universal at first, and globally recognized as a board game in and of itself.
However as we took some surveys simply around a college campus it quickly became apparent that the percentage of humans that knew the rules of chess and were capable of playing was at a surprisingly low 30% give or take 5% average for a random 40 person sample size specifically during an undergraduate college degree education or greater. These numbers are very disconcerting as in order to maximize our interest groups’ immersion the subject will need to be common and usable by nearly any human being who has ever touched a computer.

Second idea was quite honestly even worse an idea than the first, a Rubik’s cube. I have no idea why I thought of this, because not even I know how to fully solve a Rubik’s cube without hours of thought and trial and many many errors. However this brought I and my team to the conclusion of using a common object or interactive experience that people would have played with in their childhood would be the best bet.

This would correct the issue of a lot of our earlier ideas which was showing off the third dimensional depth that is the entire concept of the project, e.g. Tetris or Snake or Pacman or Space invaders etcetera. And at the end of a short meeting we agreed that our software should be unique and made from scratch while using a base idea which will be commonly understood to a large portion of the human population.

An interesting thought brought up was to stray away from the concept of an interactive experience and instead attempt to render a project subject or object being developed in AutoCAD in real time. This comes with a small country worth of problems but it is worth mentioning for perhaps animators and graphic designers to use in terms of marketability and impressive ability of the product on launch day.

In the end, two decisions were made:

1. *Investigate a program to allow Graphic Designers real time progress on their project.*
2. *Make a base “game” allowing the player to rotate and see some globally recognizable landmarks.*

At the end of the day of research I could not find any single “universal” toy or activity between all walks of life, even hide and go seek was unique to Europe and the Americas. More study is required to determine if perhaps there is a special bodily function or commonality that could be used as common ground.

### 5.4.2 Nvidia 3D Vision drivers

Our project will make use of a driver used with NVIDIA graphics cards and is most commonly used to 3D TVs. However after testing it does work perfectly fine with the 3D DLP projectors we have purchased to create our holograms.
According to the NVIDIA GeForce Experience driver update notes the 3D Vision software is already installed for any computer that is running the most up to date drivers for their GTX 1050 or better graphics card.

So we already have the software, as well as 5 different very high quality GTX graphics cards capable of running it, so hardware is not an issue right now. However there are 2 different kinds of NVIDIA 3D Vision drivers. Firstly there is the standard NVIDIA 3D Vision driver and then there is the new NVIDIA 3D Vision Discover driver.

The new Discover drivers are optimized and work with an LCD screen as well as DLP style projectors and TVs to create a stereoscopic image through their own custom made anaglyph, more commonly known as simply Red and Blue, glasses which are made with curved lens filters to maximize the 3D effect on your standard 120Hz LCD monitor or Television screen. While this makes the 3D more adaptable as well as easier to render for our poor graphics cards, it is still a step down from the original style of Nvidia 3D Vision drivers.

The original NVIDIA 3D Vision drivers used a much more advanced viewing port that is called a pair of shutter glasses. How shutter glasses work is that they are “active” shutter glasses, which contain a power source and the ability to polarize either the left lens or the right lens at any given instant. What this means is that the medium on which the 3D rendered image or images more accurately is being projected will display first a single image meant for the left or right lens to let through and then immediately after a second slightly altered image will display meant solely for the opposing lens. These images are projected for a fraction of a second and must be synchronized with the glasses in order for smooth and non “ghosting” 3D to occur.

Ghosting is a phenomenon which occurs when the human eye, even for a millisecond, sees both the left and right eye image with both their left and right eye simultaneously. When this happens the human brain tries to marry the two objects, but since they are so close and technically the same object, it appears to smear or stretch or teleport a few inches in the direction of movement the object happens to be moving at the time. Thus the object, which is usually a living being, seems to “ghost” across the screen. This is very bad and ruins 3D immersion near instantly.

In order to avoid this and keep synchronization fluid and tight, the images displayed are interrupted with a white flash or other image, that the glasses can recognize, telling the glasses to either polarize the left lens or polarize the right lens accordingly, before displaying the next image. These can also be assisted by using wireless Infrared Protocol. These signals must be rapid, and as such are far to fast for the human eye to perceive, usually microseconds or even
nanoseconds of interrupted frames just to remain barely, yet still fully, undetected in the eyes of the user.

All this being said, the drivers we must use will be the older style drivers for a higher quality as our mission statement is explicitly clear on.

5.4.3 Feasibility of Implementation

After a lot of research primarily through the google search engine I have found, as expected, that we will not be the first to create a 3D DLP ready projection using Unity as the base engine. And as it so happens there are asset packages that fully render the effect we seek using two camera objects.

This, comes with a huge problem though as the software was made in 2011. At the time of writing this paper that was six years ago and will not work even remotely well, which means I will be writing the necessary functionality from scratch, using this old asset package as a baseline to help get the proper algorithms. Starting with getting a few accurate measurements for human population such as distance between pupils and average U.S. adult height.

According to the United States medical advisory website the average male in the united states has a span of 70mm between pupils and the average female in the united states has a span of 65mm between pupils. As a start we will be setting the cameras at a distance apart relative to pupil span as a fraction of the average height of a male in the united stated which happens to be, according to the same website, 177 centimeters.

Now to compare the in game characters height, which we will now assume to be 177cm and set the camera's distance at exactly 0.0395480226 of that height, which is 70mm divided by 177cm. This ratio is critical to allow the brain to generate a single image properly, as a set of images that are too far apart will appear like it is much closer than intended, and a set of images that are much too close together will appear much further away than intended.

According to a Unity Staff Moderated blog, the plugins used by Unity are only capable of running on Nvidia hardware and therefore limits out choices if we advanced the product to a titan level graphics card. There are ways around this however, so I've looked into some blogs and reports of alternative plugins that run on the assumption of a windows 7 environment or better and no other assumption.

This is fortunate for us as we will be heavily modifying this technology to suit our needs. Research will have to give way to direct experimentation through trial and error and adjustment until the proper sought after effect is created.
This will have to wait until all the necessary supplies are readily available to perform such experiments.

5.4.4 Parsing Data from the IR cameras

This can be a little more complicated depending upon which data structure or method you are parsing form but after some researching into the depths of the Unity website I found something called JSON Serialization. JSON Serialization is very useful when we want to retrieve information from an outside source and change it to a format in which the Unity Engine would find it remotely useful. And we will be needing to do this in order to get the angle of attack measurements which will be calculated by the IR lasers and IR cameras being fed into the boards we will custom make in order to process the calculation of the angle of attack from the vertical, the horizontal, and the perpendicular dimensions’ position.

According to the Unity Engine website the exact definition of the JSON Serialization method is a feature that is used to reformat objects to and from the JSON format. Very informative I know but it does give examples to help us out. This, it goes on to say, is useful especially when interacting with web services or for simply packing and/or unpacking data to a text based form more easily and in a standardized format which the Unity Engine comes prepackaged with functions which are capable of doing just that.

The idea of a JSON Serialized format is to describe and name what data types should go into the Object or Class that is being Serialized. An example of a Serializable class is displayed below in the format of C#

```csharp
[Serializable]
public class MyClass
{
    public int level;
    public float timeElapsed;
    public string playerName;
}
```

Below this is a filled out example of the class seen above.

```csharp
MyClass myObject = new MyClass();
myObject.level = 1;
myObject.timeElapsed = 47.5f;
myObject.playerName = "Dr Charles Francis";
```

And when this object is serialized, by using the utility JsonUtility.ToJson() we arrive at a clear string of characters.
By using this, we will be able to get the necessary outside information inside the program with relative ease.

5.4.5 Limitations of the Software Vs Hardware

The DLP projectors that are used with 3D imaging software are limited by what they can take in and do, more specifically by the amount of data that can be transferred to them, the max framerate capable from the projector itself and the data produced by the best hardware inside of a desktop personal computer that a college tuition reduced, minimum wage salary can purchase.

To start off we will take a look at the projectors themselves and their specifications. The BenQ projector we have purchased is capable of projecting an image with the resolution of 720p which is the third to best possible resolution for television screens and computer monitors and the second highest resolution possible for 3d DLP projectors which are at a reasonable enough price to make this project possible. Although this projector can be fed information using Composite video feed, or a D-sub 15 pin input port, we will be using the highest quality data transfer method which currently is the High Definition Media Interface or HDMI. The maximum frame rate on the projector is 120 total frames per second. More specifically that is the two eyes added together, meaning the left eye will receive a maximum of 60 frames per second and the right eye will receive also 60 frames per second unique to that eye, so we are technically seeing the media in 120 frames per second although this is completely irrelevant because of how the brain combines the two images into one, making the actual frames per second 60 grand total.

None of this means anything if the HDMI cables aren’t able to transfer enough data quickly enough to supply the frames. The current HDMI 2.0 were rated at a maximum of 18 gigabits per second. This is translated to be a whopping 60 frames per second throughput on a 4k resolution, which is massive. At lower resolutions, the HDMI 2.0 system can put through any resolution that isn’t 4k at a max of 120 frames per second which is exactly what we need. However most HDMI cables which are being used today are not HDMI 2.0, they are actually HDMI 1.4 or below. It was only a few years ago that the HDMI 1.4b was released, with its new feature of 120 frames per second 3D output, which was not offered in any previous versions. And even then, the website lies, as what they actually meant is the frames themselves are totaling 120 frames per second, which is fine for us, but it wasn’t until HDMI 2.0 that real 120 frames per second 3D output was supported which actually means 240 frames per second grand total. Thankfully we only need the former and as such must get a cable and also a port that supports HDMI 1.4b or later data transfer protocols.
This leads into the computer itself, which has already been built by a few of our team. The motherboard is a recent model and does indeed support HDMI 2.0 and more importantly, the graphics card we have used also supports HDMI 2.0 which is good enough for our purposes, and even exceeds them in some regards. Now as this research page is serving as a sort of hardware check to make sure our project has the possibility of success, we will ask does the Unity Engine have enough power to supply the HDMI cables with the frames they need.

So how will we test this? Well this isn’t a research problem, but more of an experimental problem. Therefore as I am writing this report I am downloading software 3D demos and virtual reality 3D simulations to measure how many frames could be produced by both the Unity Engine itself and the Unity Engine when it was run on the computer system we planned to use to create the experience. I used a background output measuring tool to see the results of the tests which is specially designed to work with the NVIDIA graphics cards.

After three tests, the results were satisfactory and I was exceeding 120 frames per second almost without interruption, although the small and infrequent interruptions were concerning to me. It was decided that these interruptions needed to be ironed out due to the possibility of the causing “ghosting” or a similar phenomenon that might cause nausea. Honestly I was very surprised my system didn’t perform above and beyond the standards we set as it is a recent build and cost me a fortune. After exploring the readme instruction sets of each and every software test program I used and finding absolutely no issues in the implementation I decided to attempt the tests again, this time checking my base hardware computer's performance and processes.

The issue was immediately apparent, over half of the CPU processing time was dedicated to a process that called itself the Microsoft Compatibility Telemetry. I had no idea what this was so I looked it up and found that it was a common problem in windows 10 users, which by the way was necessary as an operating system to our current plan of accomplishing this project, and it sent a lot of data to Microsoft about my personal usage and activity while using the operating system.

After this had been taken care of through console commands and a lot of research, I ran all of the tests again. Unfortunately the results were the same as the last batch of tests, so I looked at the windows task manager to see if the Microsoft Compatibility Telemetry had come back. It hadn’t and instead a very similar process called Microsoft Compatibility had replaced it. This also was fought against with appropriate action and it was discovered this actually was a compatibility program designed to let older software run on newer hardware and operating systems such as the updated Windows 10.
I was taken aback once again as everything I had on this computer was top of the line and state of the art, yet Windows 10 had the impression that it was running old software. And this went on for hours until I found what the true issue was. My NVIDIA graphics card drivers were out of date. After updating them I ran the tests a fifth time, as I had been continuously trying to see if I had made any progress, and they went without a single stutter. This proves that both my hardware and now thankfully my software is up to the task of producing more than enough data for a steady stream to the projector.

5.4.6 Ability to use Separate Operating System

In this topic we will be exploring and detailing the many different options and if they may or may not be used to create something similar, better or worse than our current project plan. We will start with very broad and massive base decisions and work our way to more detail oriented choices. Firstly, the architecture of hardware we selected was a PC architecture. This decision was never formally expressed out loud however between all four members of this team everyone assumed we would be using a PC and not an apple computer or a third party I am not aware of. Could the same software run on an apple computer? The answer is complicated. While NVIDIA software cannot run on anything other than an NVIDIA graphics card, as well as while the fact is apple computers cannot add additional hardware such as a graphics card, there have been very high end apple computers that were able to stream a 3D movie into a DLP 3D projector with great success. However this is not what we intend to accomplish. Interactive experiences do not work very well with apple computers, even those apple computers that have been modified to run on the Windows 10 operating system. They do not possess enough power and this fact alone has made the gaming industry very very hesitant about developing gaming software for it as it would most likely not sell. All fact taken into account we must run this using a PC desktop or possibly a laptop.

The second decision was also not really a decision said aloud but rather was assumed silently by our entire team. The decision to run our software on the Windows 10 operating system was unspoken yet accepted. While we can gloss over the terrible attempts of running Mac OS on a PC desktop computer, as those attempts are only performance reducers at best and computer killers at worst, there is another operating system popular enough that someone wanting to purchase our product could assume it would be supported. The Linux operating system is a free open source minimalistic software used by approximately 1.38% of PC users according to a study done in 2015. This indicated there is a fair chance a Linux user would attempt to use our project based upon statistics.
This begs the question, would an interactive unity experience run on a Linux based desktop computer to the standards we set forth? To give a short answer, no it would not. And to give a long answer, the Linux system is certainly powerful enough to handle such an undertaking as it is quick and efficient. However despite this it has many of the same problems that OS X and Mac OS have in terms of compatibility. Unity does not like Linux, and most of the Linux users that run interactive experiences often have to spend a lot of their CPU's running time dedicated to using a plugin Microsoft legacy compatibility software to even begin the loading process.

5.5 Development Environment
For our desktop, our developers will need two pieces of development software to develop our side of things.

5.5.1 Unity
We will be using the latest version of unity available via: https://unity3d.com/get-unity/download

5.5.2 Visual Studio Community 2017
Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. We will be using it to develop the Windows side of our project. It is used to develop computer programs such as DLL’s, Windows Console Applications, .NET framework applications.

We will be using Version 15.2 of Microsoft Visual Studio Enterprise 2017 and Version 4.7 of Microsoft .NET Framework

Visual Studio provides the following benefits:
- IntelliSense: Code completion
- Code Refactoring
- Integrated Debugging
- Integrated Building
- Database schema designer
- Class Designer.

Visual studio supports 36 programming language, but we will only be using Microsoft Visual C# for our project.

We will be using a .NET Framework application to write our DLL class library.

5.5.3 C#
C# (pronounced c sharp) is a strongly typed, object oriented programming language developed by Microsoft within its .NET framework. C# is very similar
to Java and is just as easy to program in. We will be using it to write all the code which runs on our Windows desktops. This includes our DLLs and Unity Game Engine code. We will be using the latest version of C#, shipped with the latest version of Visual Studio.

Language Features:
C# is compiled, not interpreted, by the IL compiler. C# is an object oriented language, and as such it employs a lot of object oriented functionality such as:
- **Objects**: A single discrete instantiation of a structure (for which a Class is a blueprint for) is called an Object. Objects can have distinctly scoped variables and functions as members.
- **Inheritance**: A (parent) class can be inherited from if it is extended by some other (child) class.
- **No Multiple Inheritance**: A parent class can be extended by more than one child class, but a child class cannot extend multiple parent classes in C#.
- **Polymorphism**: The concept where different classes that inherit from the same parent class can be submitted as arguments for functions which accept their parent class.
- **Function Overloading**: Functions can be overloaded in C#. That is, you can define another version of a function with the same name but with different arguments in C#.
- **Namespace**: A C# namespace provides the same level of encapsulation as a Java package, where all variables and functions defined as ‘protected’ are available to any other method or class within the namespace.
- **Strong Typing**: C# supports strongly typed implicit variable declarations with the keyword ‘var’, and implicitly typed arrays with the keyword ‘new[]’ followed by a collection initializer. C# supports a string Boolean data type ‘bool’. Unlike other C like languages, C# does not allow integers to represent boolean values (0 for ‘false’, 1 for ‘true’).
- **Type Safety**: C# is more type safe than C++. Type conversions must be done explicitly, implicit conversions only happen for basic things like widening integers.

C# has syntax similar to Java. Semicolons are used to denote the end of a statement. Curly brackets are used to group a block of code together. Blocks of code are commonly grouped into functions, functions (and member variables) are grouped into classes, and classes are grouped into namespaces. Variables are assigned using a single equals sign ‘=’, and compared using two consecutive equals signs ‘==’. Square brackets are used with arrays, both to declare them and to get a value at a given index in one of them.

### 5.5.4 Source Control

An important aspect of software design is the use of source control. If a gamer was playing a video game they would want to save their progress, perhaps load
a different save where they could make different decisions, and be able to switch between their different saves. They could even merge decisions they made in one path they took into another path you took. This is the essence of source control.

5.5.4.1 Git

The software of choice for version control is Git. Developed by the Linus Torvald in the 2005. It is the most commonly used software version control system in the known universe. Git allows programmers to check their code incrementally using commits.

5.5.4.2 Github

Github is one of the most universally used remote git repository handling services. It allows programmers to handle their git repositories remotely so that multiple developers can add to them.

5.5.4.3 Source Control Workflow

The basic workflow of adding a new feature to the codebase is this:

1. Make a new branch from the current main master branch.
2. Make changes, use commits as ‘checkpoints’. Commit messages should tell a story and be informative enough so that those looking at the code later will have an adequate explanation.
3. Make a “Pull Request”, which is a request sent to the repository owner to merge new changes into the codebase.
4. Repository owner then reviews changes (additions and deletions are shown line by line).
5. The repository owner can then provide feedback and the editor can revise their code by pushing new commits.
6. Repeat step 5 until code is satisfactory then accept the pull request (sometimes it takes multiple people to accept the pull request.)

5.6 Application Design

In unity, it can be difficult to see what the end result will look like and more importantly feel like once the interactive experience is completed. That being said, you still need an end result in mind or a final project scope, which by happenstance is the exact reason we are writing a minimum page requirement documentation bundle in order to ensure that we do not deviate from our required objective during our work.

Now on the way back from class, we had a discussion about the interactive experience we should be making for the judging suits we would be presenting to. And first off, it had to be simple for many many reasons. For one thing, it
couldn't be a fully-fledged story interactive experience with a complex and compelling plot, memorable characters and an emotionally investing dialogue tree that everyone ignores because they are too busy looting bodies from a raider camp. That would take way too much time both to make but more importantly we would lose our audience. We estimated that our window of attention grabbing would be less than twenty seconds in order to show what we had done to impress the gate keepers.

The second reason it had to be simple was the learning curve. See, interactive experiences like Megaman X taught the player to play the interactive experience mechanics and way to progress through the interactive experience without needing a tutorial level, and while this would be ideal, we do not have time for that either. Additionally, and no offense to anyone, but our judging panel will not be a super professional or seasoned veteran gamers. This means that we cannot have a complex controlling mechanism to interact with our software. No simultaneous multi-handed actions.

Simple meaning easy to understand and easy to control. This leads us to the simplest objective in any video game ever. “Try not to die or at least stay alive for as long as you possibly can.” This is the ‘high score’ method of gaming objectives and is quite addicting as the replay-ability is built into the system.

5.6.1 Game structure

First we will discuss what will be in the program and what we need to build into it, starting with the most basic concepts that are common among every interactive experience and ending with the small and unique designs. First and foremost we will discuss the environment of the player’s interactive and support structure. This is also known as the floor upon which every other aspect will be made.

Before anything can be done, we must first create a new project and fill it with lots and lots of goodies to use in our interactive experience. We enter the Unity3D and click create new project. As for a destination, we will be using the documents folder as we put near everything inside that folder, do not lie you do it too. We will name this project “HoloTest_1” and save it. Now when we click the create button, it gives us a few options to include some packages in case we will need them later. These packages include, but are not limited to, 2D, 3D, particles, effects, player, vehicles and multiplayer. For this project we are going to include all of them because we have enough space on our hard drives to contain a few extra's and who knows how far we can go with this project.

Now we will take a look at how we will be using our layout which is important to understand how a Unity3D interactive experience works and how we will build our fantastic wonderland of good grades that tell large companies our life value
number. We will be going over five windows that are essential to the system process.

1. The Scene window: A scene in unity is the environment of the interactive experience itself. What the ‘creator’ camera sees. When you look at the scene you will see what elements you have in your interactive experience and where these elements are placed relative to one another. This will allow you to change objects and elements in the scene.

2. The Game window: This is a sort of test of the interactive experience itself and uses an actual player camera object to see what the user or player will see when he or she starts up your interactive experience. You can use this mode by clicking the play button in the top middle of the window and the executable will run as if it was compiled and linked onto your desktop. This can be stopped and started as often as you like but it cannot be run in the middle of a process, it must start from the beginning.

3. The Hierarchy window: This is a list and parent and child display of the objects you have added to the scene, or more accurately we have added. The main camera will always be here and by default it will be at the top of the list. You can add more objects to the scene by clicking the add object key and selecting what type of object you wish to place then you can click and drag it to put it where it needs to be.

4. The Project window: This is similar to the hierarchy window in the fact that it shows all elements of the scene, however this window displays very specifically the files that are contained within the project. This includes scripts, folders and objects if you delve into the file system of your project. This also allows you to explore your entire computer’s file system but that isn’t recommended by either Unity or this team.

5. The Inspector window: This window will allow each individual object in the scene to be changed and customized to fit the interactive experience feel and mechanics you desire to have in your interactive experience. So this is very similar to clicking the ‘peek’ button and looking at the innards of a class in the JavaScript source files.

Now that we understand how to start up our build in Unity3D and now that we have a folder filled with prefabricated helper tools and sample textures to make a sort of good looking interactive experience. Let’s start slow by placing down a completely flat as well as a very large surface and name it “ground” so we can keep track of what object is what when we delve into the source code to fix the inevitable bugs. This floor is actually a cube object that we flatten and extend outward to create our floor, this is the easiest solution for our purposes as an
actual “wall” is something that shouldn't be used as a floor, as it is prone to chinks in the armor around hills and other terrain.

From there we will add the most important object in the entire interactive experience. The player. The player object can take any shape, any size, and even any texture but they all have a few things in common. First and foremost get a player object from that player prefabricated tool we compiled this project with at the beginning because it has something essential which is the main camera. The second thing this object has is something called a “rigid body”.

Rigid bodies are what we call the things in interactive experiences that you can see and everything except for particle effects have them. These things allow us to actually see the player when we look at it from another camera's perspective. Now you do not necessarily have to leave it with this rigid body as some interactive experiences are entirely singleplayer much like the project we will be creating here, but for development purposes you really need it and will use it or suffer the consequences. The third thing is a really important thing too, which is the “collider”.

Unlike the rigid body, which is used for interacting with the cameras and only for interacting with the cameras, the collider is what allows the object to touch and push back, as well as be pushed back against, the other objects and environmental set pieces in the scene. Without this collider you would become nothing more than a ghost, floating in the scene and pass through walls and while this sound fun in real life, in the interactive experience world the entire point is to be challenged so we will leave our player on his two feet for now, until we turn him into a rocket ship or something which will force us to adjust our collider, but not our rigid body because remember we really do not care if that looks good unless it is from a third-person perspective, because the player will never ever see it unless it is third person. We will revisit this topic later.

5.6.2 Game components and design goals.

In this topic we will be explaining precisely what the first interactive experience program will consist of and how we will implement those components in our chosen engine. Firstly, we will be using of course the 3D cameraworks plugin as the 3D portion of this is the entire point of this project. Our interactive experience's goal will be to survive as long as possible, and force the player to move into a path of danger or create an environment in which danger is unavoidable without cheating or stopping the process in its entirety.

The interactive program will require a menu selection screen upon boot that will allow the user to select from a few different option including but not limited to: Play game new, Options, Reset records and highscores. As an incentive to do well in these interactive experiences, a small database will be set up to record
and track whom has the highest scores in our particular interactive experience when we release it in case multiple users are present and wish to compete. The scores will be based entirely upon the grand total time survived in the interactive experience environment as it will be the only goal and only reward upon completion. How we will achieve this environment is very simple.

Most interactive experiences have the player controlling as many aspects of their character's movements and actions in order to more fully immerse the player inside the virtual fantasy world they have created. This is something we cannot do as it doesn't promote very fast gameplay and want for a subsequent trial run in order to obtain a better score. The player in our interactive experience will only have 2 controls, tilt the direction of movement to the right and tilt the direction of movement to the left. The direction of movement will be constant and continuous without any input from the player.

As the player, most likely modeled after a rocket ship or similar, will control their survival by avoiding obstacles which they cannot help but speed towards as we have removed their ability to slow down or stop. There are two ways that are apparent in order to implement this. The first option is to make the player's position static in the level, and have the level itself move toward the player and the players controls will change how the level advances instead of the player's character itself. However from the player's perspective this is interpreted as movement originating from the player so the effect is not lost.

The second option is to have a static level or ground and to move the player and the camera object through the level, which is the much more common approach and also the approach we plan on taking to implement this goal. The static level will consist of either randomly generated blocks or of predetermined blocks so as to not create an unfair environment for the pursuit of success. The issue with this idea is that not only do we need to move the player object, but if we wish for the experience to be essentially endless then we need to add more level continuously, as well as necessitate the random generation process due to the time constraint of I do not have infinite time to create this project.

If we are going to randomly generate the oncoming level, how often will we update the level? At what distance will the player be able to see the oncoming blocks of death? Well distance is all relative to size of the player and the speed of the player so there isn't a magic number we can just research to get a perfect solution, instead we think it would be best to determine the reaction time or time of processing events or situational evaluation time in order to determine a proper speed and visible distance meter. In order to determine this value we first have to create the interactive experience nearly in its entirety and then play it over and over and over again, essentially becoming our own alpha testers. Then we will adjust the values for speed and visible distance radius over and
over again until we feel that it is a fair margin of expectations of our player or user.

We mentioned a high score system to encourage players to keep playing and to try their best to succeed, but we didn't discuss how one would set up a database to hold these scores. As well as other questions such as how will unity get those scores? How will the scores be accessed in interactive experience? What computer will sort the data to find which score is indeed the highest? This can get rather complicated and to start we will need to set up our own website which will house the data.

Nothing too fancy is needed, in face we do not even need it to be complicated in terms of what it sends and how large the website is, as we would only need a few kilobytes of space in order to store usernames and high scores. First thing to do is set up a free website in a domain temporarily because we are poor and do not feel like spending our college tuition on a fancy website. For this reason we are going to use the free website hosting service of 000hostwebsite.com and their free month trial system. In the meantime we can use a virtual or fake database communication software to test out its functionality and operability. Afterwards we can implement the real database hosting service and three weeks should be more than enough time to work out any bugs that arise that were not previously found and dealt with.

Unity doesn't have any prebuilt support for a connected database and generally it isn't advised to use it. According to a number of professional blogs the best way to save progress and high scores is to make an additional plugin using SQL and C# based functionality. This makes a system interact automatically with another system incredibly difficult. In order to truly get a window into database connection we might need to fool the computer into thinking that it is populated by keyboard commands when a keyboard doesn't currently exist.

See, when we put a USB male end device into a female entrance USB port, the computer we are using doesn't care what that USB is connected to, what that USB may or may not have gone through to come to this state, no, all it cares about are the ones and zeros it conducts through the metal electric conductive interface that it gathers and then obeys depending upon what its particular operating system decides should be done with those instructions. In our case we will be using windows 10. So if we set up our interactive experience keeping in mind that there will be two sets of control scheme inputs, one being used by the player in order to influence the virtual world and one influenced by the website or database, we can map the inputs using the control object found in Unity3D's prebaked tool kit, though we will have to custom make our own keyboard interface drivers.
We plan to rewrite the USB device in order to send updated signals to the stereoscopic 3D output minimum once every second, and if the USB has not received any new information it will simply send the previous packet of data so that we can program the interactive experience keeping in mind it will believe it will always get the data every single time and it can adjust accordingly.

Crazily enough, we can do almost precisely the same thing with our database, through the interface will be a little more complicated. When we write up a program to automatically access the database, instead of having it send a signal to a physical input, instead we can make the computer think that there is another physical input when there isn’t one. Essentially a virtual USB driver type object that will send the strings of data to what appears to be a third method of control, but we all know here in the real world that there is only one method of control, namely the controller itself. This does mean that what we create will work with a near separate display whenever the highscores are generated so in terms of keeping the system in a logical and self-contained folder, this portion will seem more of a vestigial organ, which is we suppose lucky in the event of its failure the rest of the project will continue regardless and without care, while it will throw an exception, we will force the computer to ignore that particular exception code if thrown by that particular class type.

There is another option, however, which is the JSON format class type transferal. This is a legitimate technique, but not one which would be very time efficient, as writing a driver for converting a normal batch of integers or doubles into a string and then converting that into the JSON format and then sending it to unity in order to update every single frame at minimum 60 frames per second is just too long of a process and involves too many conversions to be made a viable method especially in the limited time we have to complete this project.

5.6.3 Game Description

The first screen the player will be brought to upon powering up the system and running the executable file we will be calling Holo-Run is going to be the main menu. From this main menu the player will have a number of options, listed before them vertically and will include, but the options are not limited to, the following list: “Play new game.”, “View Highscores.”, “View Options.”, “Exit Game.” As well as “Resume Paused Game.”

We will explore these options from most important to least important paragraph by paragraph. By selecting the “Play New Game.” Button, the interactive experience will load a starting point zero interactive experience and immediately begin by creating the plane of blocks and the player, giving the player a countdown of three seconds before starting a momentum build
forward until a constant speed is reached. Once this happens there is no stopping the player object unless you either die or the user has a heart attack.

This doesn't include the calibration method that are necessary in order to actually play the interactive experience, and those will come either before or during the very first playthrough of the interactive experience where the player will be asked if they have or have not gone through the calibration process.

The interactive experience will continue to go at this constant speed, with the interactive experience not ending until either the interactive experience is paused or the player fails to avoid any block inn his path and dies due to a wreck. At the time of destruction a particle effect will play to indicate a negative consequence of the player's actions and encourage them to make sure that doesn't happen as often as they can help it on a repeated playthrough. Then the screen will fade to a menu screen indicating their score and whether that score was a new record highscore for the particular player and another button that will reset the stage and level if they would like to try again, or another button indicating a return to menu.

If you select the resetting button which will be displayed as a large off colored button with the text “Play again?” On it to entice the user, you will play again and that button will not actually be any different than the main menu button named “Play New Game.” That being said, if the player selects the menu pathway button which will be neutrally gray colored and be titled with the text “return to menu.” It will do just as the button suggests and will load the menu, however what actually happens is the interactive experience will basically redo the load of the executable.

The final size of this file will be relatively small compared to some of the other files it can load, and according to small scale tests the loading of the executable into the ram and starting up the process will take just as much time as repathing the entire system of objects and classes so that we can return to the main menu, so we will just save a little bit of time by simply launching the executable in order to return to the menu.

Upon returning to the main menu, or in our case reloading, we come back to the four or more options we can choose from to continue our interactive experience with holo-run. Next we will assume that our player will select the “View HighScores” button at the menu in order to view his latest victory. On the screen following this button press, the database server will be read and updated n his particular computer, and if an internet connection is not possible only his own scores will be displayed. In the case of no internet being available to check scores, the user’s personal scored will be sorted and ranked by the program itself.
There will only be one button on the high score screen and that will be the “return to menu.” Button. As expected this does what it implies and brings us back to the player’s menu and is a Segway into the last two options of the menu screen. “Exit game.” And “View Options.” Clicking the view options button will bring us to a few settings we can change about the interactive experience, namely difficulty, and volume settings. This is pretty standard so I will not go too into it. Which brings us back to the last option which you will find if you hit the ever present “return to menu.” Button you will see the exit interactive experience button. This doesn’t nothing but shut down the interactive experience and stop its running.

5.6.4 Multiplayer Design

Multiplayer interactive experiences are a natural development of human interaction, as humans are social creatures. Given enough time any franchise will evolve into a medium comprised of an online community even if that particular franchise is a single player only style interactive experience IP. Online forums, community chats and other system will lead into a social enterprise. But actual interactive experience to interactive experience and player to player real time interaction and competition, whether cooperative or competitive, is the future of video games in the coming generation.

For the purposes of this discussion, let us assume that we have mastered the art of the stereoscopic camera object in Unity3D and can use it just like any other object in that we can make multiple copies, make parent and child object and other fundamental operations in object oriented programming. How would we create a multiplayer interactive experience?

This first question is pertaining to the existence of a multiplayer interactive experience in its feasibility only and asks whether it is even possible before we proceed to ask if it is what we should do. Does Unity3D engine have the backbone to create a multiplayer interactive experience for a 3D virtual reality experience such as what we are creating? The short answer is yes it does but we would need some adjusting to the standard way to go about it.

Unity3D has a pre-built class, or rather a multitude of prebuilt classes, that assist in the multiplayer aspects of interactive experience creation. Notice I said assist, as this is not a plug and play style of software and take a lot of fine tuning as each individual interactive experience functions radically differently when multiplayer interaction is involved. For example there is a world of difference between the multiplayer interaction in a Facebook interactive experience and the multiplayer interaction in a class based first person team shooter such as Overwatch or Team Fortress 2.
Now, one of the team members in the Holo-Run project has already attempted to create a multiplayer interactive experience inside of the unity engine and succeeded with very little past that. So we can say with confidence that creating a single player interactive experience and then making a multiplayer add on to that single player interactive experience is nearly impossible. When it was last attempted the entire project had to be rewritten nearly from scratch in order to get all the parts to work with each other, so this process isn’t to be taken lightly it needs to be decided early on and stuck with. The reason a lot of interactive experiences in the current generation require an internet connection, even in non multiplayer modes, is because they use a design choice that essentially makes all interactive experience modes a multiplayer mode but instead of playing with other players it only spawns one player who is also the host player, which is the current user. This is much much easier than creating an environment solely designed for offline play and then going back and creating an entire second environment solely designed for online play.

The first object we will need to create will be an object called the network manager. The network manager is responsible for keeping the prefab object in unity called “Network Manager.” Every single script that needs to be in a multiplayer interactive experience is included in this folder, beginning with the spawning of the host player or any player for that matter. To do this we will get the player controller script and apply it to the rigid body we call a player to make a solidified player object. Once this is done we will give this object to the player spawner in the network manager to give it which object needs to be spawned in the scene. Unity will not accept this and will refuse to spawn such an object. Why does this happen?

This happens because every single object that is inside the scene, which by the way is created and run by the server and not by the players individually, needs to have something called a network identity. A network identity keeps track of the object and relays crucial information such as its unique existence as well as its movement, its player interactions and its environmental interactions. So to give it the ability to be on the network there is a prefab object component that we can add to the player object in order to give it the ability to be synchronized across the entire server. This means that if one player called player 1 spawns in and takes one step forward, it will take one step forward on every player’s screen as well at the same time.

This also brings up problems of smoothing and latency issues but we will discuss those later in the topic. This should be all we need right? Well if you build your executable file by using the Unity3D build tool and build for windows and mac and linux it will be able to host one server with a interactive experience running in one window and then run the interactive experience and join as a client in another window, however you’ll see that there is only one
player object on your screen and in the map. This is not true there are indeed two objects in the scene but two things are going wrong.

For one thing the two players are spawned in the exact same x, y and z coordinates so they are overlapping each other, the other thing is that there is no collision aversion between players yet so the interactive experience thinks that it is perfectly ok for two players to meld within each other into the ether and that will not break several Newtonian laws of the impossibility of a room temperature supersaturated solid. This is of course wrong and we need to fix both issues before proceeding, starting with the collision. Thankfully this is quite simple.

Go back to your prefab called player and take a look at its components. There isn't anything there for collision yet so we will right-click the list and choose to add a rigid body collider as we discussed beforehand. The default will be a sphere of some kind or if you're using an older version of unity it will be a sizable cube. We could make it so that it fits the shape of the player but there isn't any need for development purposes and if it becomes a problem it can be easily changed and adjusted so as to solve any interactive experience breaking bugs or exploits.

If you go back and try to build and test again, it will still spawn in the same spot but one of the players will be launched in a random direction because of the physics engine knowing something is deathly wrong with the world in the first frame of the spawning process. We do not want this to happen as is expected. We need more places for our players to exist at the start of each interactive experience, so we will create our own new spawn points to do just that. Create an empty object and rename it spawn point for clarity's sake. You'll notice just like every object you've ever created there are coordinates for where it is in the scene in components already even though you specified it to be an empty object. This is a feature of Unity3D in which it keeps good track of literally every single object ever spawned, even the ground, the sky and the wind particle effect. This makes it much easier to click and drag the spawn point object over to some place other than directly on top of the first object and you'll notice that the coordinates, which the Unity3D engine calls its 'Transform' which by the way is itself a complete and total object, though much more resembling a C style structure typename, adjusts accordingly and is changing to fit whichever location you place it.

When you do this, you have to add a component to the spawn point named “Network Start Position.” Which is also a prefabicated object that gives the network the information it needs to use the spawner accordingly. Add the spawn point information to your network manager and take a look at some of the options very near where you added the spawn point. There is an option to change the method the server decides to choose which player is manifested in
which spawn point transform. By default this option is set to Random. This is also bad as this means the possibility of two players spawning in the same spawn point transform is fifty percent which is unacceptable. The option we will be using is the second selection down the list and is named “Round Robin.” This is perfect for our purposes because it will look at all the spawn points in a specific order and see whether or not the spawn point has already been used to create a player object, when it finds an empty one it will immediately use it and ensure that no two players will be spawned on top of one another unless the number of players in a single interactive experience or server exceeds the number of spawn points available to the spawner in the network manager.

This is networked and the players are indeed in the same server, however we haven't told the server anything about how to synchronize the movements and actions of the players involved. First step to this is to define the “local player” so which one of the player objects in this server is my player? For this we designate the network identity of the first tank and we set it to be the “local authority” so that the main camera that my camera is running only looks at the one player and to do this we must go back into the controlling script and use an if statement that says if this instance of the player is not the local player instance then do not send any keyboard input to the other players. An easier way to do this is to determine if the instance is the local authority and if not we will delete the controller script from the object entirely.

Now we will fix the next inevitable issue, the fact that we cannot see the other player’s or the client’s movement or actions. This means we need to synchronize our transforms and here we have another component called the network transform. The network transform’s only job is to keep the information of the position and facing direction of the object and at a set time or number of times per second to set the maps of all the players environment simultaneously so that everyone can see and interact with the exact same visuals.

And yes there are indeed the exact same visuals, but if you run your interactive experience and you turn one of your players, on the screen you’ll notice a lot of lag and latency when it comes to the other players actually seeing the other player move around its Y-axis. The fix for this is simple, we go back to the Network Transform and down to that classes sub option, the rotation portion of the transform. Set this interpolation rate to 15, or in layman’s terms, very quickly, but do not go higher than 15. If you go higher than 15 rate of interpolation then slow moving objects will look very choppy and will destroy our visual immersion as much as a poorly synchronized player movement would have done.

Another way to run it a little bit better is to set your rotation data sending method to ‘compressed’ so that less data needs to be sent. The next thing to do to improve our interactive experience would be to start creating some
improvements to the network manager. Right now our network manager is working as a predefined basic and minimum required possible network manager to be considered a multiplayer interactive experience. However there is so much more potential in something like this just waiting to be unlocked. For example let us consider the following.

What if we were to create a script that could control our interactive experience in a more unique way? Perhaps we could add a camera that would turn off when the player began the interactive experience and then after that player’s death the camera would turn back on and link itself to the death player object. This is a very common application of gameplay interaction called a lobby spectator camera and you will find it in almost every single interactive experience imaginable however it’s a little more complicated than the basic Unity3D engine prefabricated scripts would have you believe so we will need to write our own in order to get the desired effect.

Enter the networking manager and select the script named networking manager_camera control. You'll notice it is not a monobehaviour and it is not a networking behaviour, which is the most common base class for nearly every object in the entire interactive experience you create, but instead it extends the class Networking Manager. The reason we extend this class is to gain access to methods such as OnStartClient(), which will automatically run whatever is put into it upon a client accessing and entering a server, OnStartHost(), which will do the same only it will wait until the host uniquely joins the server and begins it, OnStopClient(), which will run upon the disconnect of a player that is not the host, and OnStopHost(), which will run upon the disconnect and therefore destruction of the current hosted lobby and server protocols and database connection.

In order to create a lobby camera to see the level before the interactive experience begins, create a field for the scene camera that we will use to send transform data to our lobby camera. Then create camera rotation value field, specifically one for the radius of rotation, one for the speed of rotation, and one for if we want it to rotate at the current time.

Inside of the OnStartClient() method you can set the boolean value of if the camera can rotate to false as we do not need anymore lobby camera, someone has started to play the interactive experience and is controlling their own camera therefore we do not need to give them something shiny to look at in the meantime while they are looking at the lobby menu.

We can do the same thing but in reverse when looking at the OnStopClient() method. When this method is triggered we know that on the particular computer that this method was run, unless for some reason this player is running two instances of the interactive experience which is very difficult and
also kind of pointless, there are no players in their running interactive experience that are playing the interactive experience so we can set that boolean can rotate value back to true.

We will not get into specifics on exactly how to rotate your camera in your particular interactive experience as each and every interactive experience is unique and special (I hope for your sake). However inside of your update method you can write a checking line of an if statement to see if the boolean value is true or not for if we can rotate. If we can, let it pass but if we cannot just say return. After this line we can calculate the rotating radius and speed depending upon which level of the interactive experience we are looking at right now and then start rotating.

Now we do not need to create an entirely new camera object for this, as is common practice in interactive experience programming, there is only ever one camera much like there is only ever one screen to view the interactive experience, therefore all we need to change is what controls the camera and how it controls the camera. We can click and drag our new network controller_camera controller script to overwrite the current network manager script but keeps all of the different components from the last version while adding the new components from what you have just written.

But how will we match other players to our current player in order to experience a interactive experience playing alongside other players? This is something we need to implement which we who work with Unity3D call a lobby system or a matchmaking system. The lobby of a interactive experience is the interface for a player to examine the current running servers for a particular interactive experience and usually go into one of the servers based upon the information given by the lobby to the player.

These snippets of information about the server are typically things like ping, or latency of the particular server, where the particular server is located, what map the server is running, what interactive experience type the server is running, how many players are inside the server already, how long has the server been running this current interactive experience, what is the maximum number of players that can be inside of the server and playing the interactive experience simultaneously, does this server allow spectators on top of that maximum number of players.

At this point we are going to need to save the exact scene as we have it into our documents folder, to keep it nice and clean and ready for use when we implement our lobby system and take what we need from the scripts we have been using. This is an important step as if you do not do this all the work on the scene you have done up to this point is useless and you cannot recover much
from the scene besides the prefabricated additions you have made to your prefabs folder in the project documents.

The file you just saved will be used as our LAN interactive experience, and will act as a singleplayer interactive experience version while still being networked so that no matter what happens the interactive experience can still be run without a constant internet connection. This ability will be disrupted when we continue with the following steps in our interactive experience creation methodology. It is important that we keep in mind how dangerous a lot of these changes coming up are going to be so it is heavily advised to save often in case of a crash or a failure of permanent proportions.

We will now create an entirely new scene and add it to our interactive experience. This is a completely blank scene and we do not need anything fancy in order to come up with a lobby and matchmaking window so just save as in your project folder and label it as your lobby scene. When we do this, go to your build settings and take the lobby scene and place it into the build list first, this is very important. And then go and put in your interactive experience network manager scene as well into the build settings and hit build.

In order to give our players the ability to match up in a lobby before and after each interactive experience is completed, we will be using and modifying a prefabricated Unity3D made script which is called the Lobby Manager script. Go ahead and add the network manager HUD script so that we can more easily go into our interactive experience and test out a lot of the functionality.

We will need this functionality in order to give the player the experience necessary for a quality gameplay experience. We call it a necessity but it truly doesn't cover it. For a great look at a well functioning lobby system as well as an interactive experience browser and server selector, take a look at the free to play interactive experience called Dirty bomb. This is a personal favorite of one of the team members and it showcases just how streamlined a quality lobby can be accomplished.

Now, in order for this to do what it needs to do, we need to give the lobby an idea of what a player is, but not only that we need to give it the prefabricated object called the lobby player, which is a spawned and interactive player who is not playing the interactive experience yet but is still inside of our lobby.

We make this object in case we want to give the player the ability to chat with other players or change some of their settings for the match or move to a different team or something along those very vague lines, which we will use in order to entertain the player for the time before a match while the network manager is busy getting each and every other player connected and settled in order to begin the multiplayer interactive experience.
The first thing we will do is create another empty game object in our current scene. It is an object without nothing but a transform and that is all, so we will add a component to it. This component is the network lobby player component. All this object will do is be a placeholder for our players and ask their computers if they want to see the lobby screen and what interactive buttons they will have access to and how they want to perceive the experience as it stands.

Now what is my lobby player? This is my lobby player class which we just created in our network. You will see a few options for lobby player, game player, lobby scene and game scene. Unity3D knows the concepts of the game lobby already but it needs a little help in knowing what to spawn and when to spawn it.

So set your saved scene that you just put in your file explorer subdivision documents and your project folder and place it into this appropriate slot so the computer knows what to spawn in order to begin the lobby for players to congregate in. Next we have to give it the actual interactive experience scene when the players come in and the interactive experience begins. This is where we will put the appropriate prefabricated classes so Unity can handle it according to its subroutines.

There are a near infinite number of ways that an interactive experience can be designed for a group of players. The more players you have the more complex the interactive experience becomes and the more difficult it is to control the difficulty curve. The most popular of the gaming multiplayer platform systems will be discussed and evaluated.

The most basic multiplayer and also the first style of multiplayer is really not multiplayer at all. The “Facebook multiplayer” is really a single player experience while you can easily inform other players of the same interactive experience of your progress and in some cases send those predeveloped items or boosters in order to temporarily accelerate a friend's progress as the only form of player interaction. This type of interactive experience design is a financial decision only and it is not satisfying or fun. It is an insult to anyone who cares about the cultural and developmental importance of video games in the areas of entertainment, education, training and social mingling to even suggest the idea of a “Facebook multiplayer” interactive experience design as a legitimate decision.

The second form can take two styles, competitive tactical and competitive isolated. This style is called the “Race multiplayer” and is the most likely form of multiplayer we would use in our own project, Holo-Run. Competitive isolated is not strictly speaking a multiplayer unless there is some form of communication
between players during play. How it works is two players, typically in an interactive experience based around getting to a finish line fast or surviving for the longest time, will begin the same exact course at the same time and at the same speed. The players cannot do anything to win except drive or play their best and hope that it is better than the opposing player, whom they can see at the same time they see their own character.

Competitive tactical is much more engaging as it is nearly the same style except both players are racing or playing on the same map and course and can interact with each other, providing opportunity to assist or hinder the opposing player based upon decisions made in the race such as running into the other player, taking power ups before they have a chance or using those power ups to slow or stop the opposing player’s progress. The wonderful thing about this style is that there really isn’t a limit to the number of players that can be put in the same race at the same time except for environmental to player size ratios and chaos theory.

Now we delve into more recent styles in today’s gaming industry which is much more biased toward the first person shooter style of interactive experience design. Instead of different forms of the Tactical first person shooter, it is more akin to a sliding scale of tactical cooperation. The first to display this was the Call-Of-Duty interactive experiences with their style of team vs team deathmatch. In this way, each player has access to the same weapons and opportunities to defeat enemy players, and therefore is less a team cooperative effort and most players will go off on their own to attempt to defeat as many individuals of the opposing team as possible, because the only way to gain an advantage in numbers would be to shoot your team mate’s attacker before he or she has a chance to defeat your teammate.

Communication isn’t really necessary to be good at the interactive experience and while there are many players it isn’t very social and the skill cap is very low. As a side note, a skill cap in an interactive experience is the level of skill a player can achieve before further practice cannot make that player become more effective at winning the interactive experience. It’s very difficult to get a handle on the exact methods a developer would incorporate such a system.

In order to almost force cooperation, interactive experiences created what are called “Class based shooters.” Class based shooters are very similar to the call-of-duty style of gameplay except for the class differences. Instead of giving every player exactly the same opportunity to bring different ways to defeat the enemy, none of them are unique among players. This is where classes come in. A class is a type of player whom can assist his team mates by either improving their chances or ability to defeat the enemy or can more effectively do so himself or herself with the help of his or her team. The key assumption here is that while at first class based shooters tended to let the different classes be
replicated on a team multiple times, in recent years it has become practice to force teamwork by allowing only one of each unique class to be used by a single team. This encourages communication and coordinated effort and is most well represented in a interactive experience called Rainbow Six Siege, in which the lack of communication is considered a pre-game indicator of a severe increase in the chance of failure. And the uniqueness of each class has become even more pronounced with the rise of Overwatch.

Out of all these options, the easiest to implement for our interactive experience concept would be the Race multiplayer, competitive isolated. And the most fun and engaging case would be the Race multiplayer, competitive tactical.

5.7 Limitations on the Design

While a great attempt. There are effects created that prevent the system from being indistinguishable from real life. The first is the eyes’ focusing function. When a perceiver’s eye looks a object that is close, objects that are far away become blurry. This is due to the eye changing its focus to the closer object, while the further object then becomes out of focus. Likewise the reverse is true. When the eye looks at a further object, the closer object is removed from focus. This effect cannot be easily simulated in our system. Most importantly, it requires knowledge of what the perceiver's eye is currently focusing on. While research has been done on this effect for VR such as foveated imaging, we chose not to simulate this effect in our system for difficulty, and the overall added effect would be subtle. Instead, user's eyes will focus on either the display itself or attempt to focus on the virtual environment. If the user focuses on the display surface while perceiving the objects, the object's will remain clear, but the user's sense of depth may be affected since the focus which makes a virtual object clear is a smaller depth than the stereoscopic depth. The more natural method is attempting to focus on the object's in the environment. However, when the perceiver does this they are removing the display slightly out of focus. So the overall effect might make the system slightly blurry.

The next limitation is the lighting. When we use projectors, we project light onto a material which diffuses the light for viewing at any position. However, if a black color needs to be shown, all external natural light must be absorbed. This causes a conflict where the material needs to be a dark material for the dark colors, otherwise external lighting would render the color on the material a lighter. However, the material needs to be light enough to diffuse the projection. In our design we chose a grayish material to accomplish this. But this means we do not have the color resolution of most LCD screens. This is the case with most projectors.

With this, we cannot render perfect lighting conditions of the real world and the virtual environment for realism. In the real world, there are many other
variables contributing to an exact color (light intensity and frequencies). In addition, we would have to match the external lighting conditions. All this being said, our system was not designed for realism in terms of the object appearance, only realism in the projection and depth. Most of our applications are stylized differently than the real world, whenever for artistic or clarity.

Even if we know exactly where the user’s head was in 3D space, we have another inaccuracy. Due to different eye positions and head shapes in humans, the overall eye position might be off. For most user’s this should not matter too much. Most users should be close to an average eye position. However, for user’s whose eye position differs largely from the average, they might experience a slightly offset effect. The assumption for our project that this configurable and is adjusted on a per-user basis. However, this relies on the accuracy of the user during calibration. Therefore, there may be some inaccuracy, but should not prevent the overall effect, and still give most users a good experience.

Overall, the system is designed to have a small a latency as possible. That being said, one of the biggest latencies is the input lag of the projector. This will more than likely be up to 20 ms which is significant. To compensate for this large lag, we attempt to perform some prediction as mentioned in the head tracking algorithm. Nevertheless, this is still a tradeoff between the accuracy of the prediction and the overall latency of the head-tracking system. The overall temporal-space accuracy is still not perfect, and may be detectable for some users.

Finally, one last limitation is the ability to create objects that appear outside the display. When objects appear to pop out of the display, then a least a portion of all perspectives cannot be rendered correctly since the rendered object will extend past the display shape. Specifically, when viewed from a position that is on the display shape’s plane. Because of this, we can only simulate virtual objects inside the display, or some objects which are not rendered off screen from most perspectives, being shortly extended from the virtual environment, and/or small in size.
6. Testing

In this section we discuss our various methods of testing the design in both future and early stages.

6.1 Software Testing

6.1.1 Unit Testing

Unit testing is an important part of developing proper code. A unit test is a test to make sure expected functionality is carried out across a single function. Unit tests often are developed so that they all automatically run during the execution of the build process, in which the software project is compiled. Unit tests run and can be configured to throw exceptions, log warnings, or halt building when they fail. This allows developers to have peace of mind when developing large software projects with many interconnected parts. This also ensures that when a programming changes a function in one part of the code, it does not cause unintended effects on other parts of code.

Visual Studio has built in templates for developing unit tests for the code called Unit Test Project. It allows programmers to test their software at the function granularity.

6.1.2 Integration Testing

Integration Testing are conducted towards the end of the project life cycle, where many different pieces are sewn together. They involve testing the system using multiple components, they are often more complicated to enact than unit tests, and may sometimes need external software to test.

For example, in order to test our functionality of our Serial Reading DLL we need to use a virtual Serial Port software called Null-modem emulator.
Using this and a short python script to write to COM5, we can read COM4 and test our software in a more end-to-end fashion.

6.2 Hardware Testing

In this section we are highlighting the test plans and procedures for the hardware portion of our project. The hardware tests are going to be split into the following categories:

1. Power Supply Circuit
2. Battery
3. DLP-Link Communication
4. IR LED Communication
5. LCL Operation
6.2.1 Component Hardware Testing

6.2.1.1 Power Supply Circuit

The Power Supply circuit test plan was pretty straightforward and easy to enact. We would simply connect the micro-USB voltage source to the circuit portion including the power diode and the Vin of the PMU and see whether the PMU is powered up and the LED lights up correctly. If that works correctly our second test will be to make sure the battery is recharged properly when plugged into the power source. We will measure the amperage of the battery before and after to make sure the battery is being recharged.

6.2.1.2 Battery

The battery test will be very straightforward and easy to test. The first test we must figure out the input voltage to verify that the battery has a sufficient amount of power being sent to the device for it to run and also make sure we are not sending a surplus of power that could harm any of our hardware. After our first initial test for voltage we would test the battery again for its output voltage after a full recharge to make sure that it is charging successfully and completely. We will be testing battery life by running the battery at the rate we would be using it when we run the unit as a whole, and checking if it runs long enough to meet our standards. We will also be checking to see if it’s recharging and lasting as long for successive runs. The last major check we have is a max load temperature check where we see what the temperature of the battery would be if all components are set to their max load. This check would strictly be for safety purposes to see if the battery can handle the specified loads without becoming dangerous to the surrounding components or even us.

6.2.1.3 DLP-Link Communication

The test for the DLP-Link connection will be simple and it involves only two steps. The first portion of the test will be making sure the sensor receives any type of signal. To do this we will set the projector to send out the DLP-Link and we will check to make sure the sensor picks up the white light signals. If this portion is successful we will then test the sensor in conjunction with the filter and MCU to make sure we are receiving the correct final digital sync signal.

6.2.1.4 IR LED Communication

The process of testing the IR LED communication with the IR Camera will start with us simply testing the IR LED for correct functionality. Once we see that the LED works we can move on to the next portion of our test, testing the functionality of the IR Camera. We will do this by again activating the IR LED and this time making sure to see if the Camera picks up the signal that is being sent.
If it is capable of that step our last portion of the test will be to check if the Camera can pick up the entirety of the output signal being sent through the IR LED. We will generate some sort of coded signal and see if the IR Camera will be able to pick up that signal and use the coded data.

6.2.1.5 Liquid Crystal Lense Operation

Testing the LCLs will be minimal. The first thing we have to check for is correct operation so we will supply a 10V source to the lens to make sure it becomes opaque. Once we see that it operates correctly we will supply a signal at a 60 Hz frequency to each individual lense to check that it is shuttering correctly. Our final test will be to connect the lenses up completely and ensure the lenses work under normal operating conditions where both lenses shutter opposite and at a combined 120 Hz.

6.2.2 Hardware Test Procedures

6.2.2.1 Power Supply Circuit Test Procedure

Below is the table giving a descriptive step-by-step test procedure for testing our power supply circuit. The test procedure is based off of the test plan we have written in section 6.2.1.1.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Conditions</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The PMU and the Power ON/OFF LED will be tested for good functionality</td>
<td>The circuit will be tested using a Micro USB compatible power source under normal operating conditions</td>
<td>The PMU should receive power and the LED should turn on correctly signifying power ON</td>
</tr>
<tr>
<td>2</td>
<td>The Circuit will then be tested connected to the battery source to make sure it can recharge the battery without issue when connected to the power source.</td>
<td>The circuit will be tested using a Micro USB compatible power source under normal operating conditions</td>
<td>The circuit should resupply the battery with power and the measured amperage should be higher after charging state</td>
</tr>
</tbody>
</table>

Table 10: Power supply circuit test procedure
6.2.2.2 Battery Test Procedure

Below is the table giving a descriptive step-by-step test procedure for testing our battery. The test procedure is based off of the test plan we have written in section 6.2.1.2

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Conditions</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The battery will be tested at a full charge for its output voltage.</td>
<td>The battery will be tested under normal operating conditions with an average multimeter.</td>
<td>The battery should be outputting the stated voltage</td>
</tr>
<tr>
<td>2</td>
<td>The battery, with a full charge, will be continuously run under normal conditions to test the battery life of the system.</td>
<td>The battery will be tested under normal operating conditions again until the battery dies.</td>
<td>The battery should be able to work within the minimum power requirements for our suggested run time</td>
</tr>
<tr>
<td>3</td>
<td>All of the components will be run at max power draw to test the max temperature that the battery reaches. This serves as a safety test.</td>
<td>The battery will be tested in a safe location under max load conditions for all the components.</td>
<td>The battery should be able to run without reaching dangerous temperatures.</td>
</tr>
</tbody>
</table>

Table 11: Battery test procedure

6.2.2.3 DLP-Link Communication Test Procedure

Below is the table giving a descriptive step-by-step test procedure for testing our DLP-Link Communication. The test procedure is based off of the test plan we have written in section 6.2.1.3.
### Table 12: DLP-Link test procedure

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Condition</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The DLP-Link sensor will be tested for correct functionality by exposing it to the white visible light</td>
<td>The test will be conducted under basic operating procedure of the sensor</td>
<td>The control unit will receive the signal transmitted to the optical sensor</td>
</tr>
<tr>
<td>2</td>
<td>The sensor will be sent signals from the display unit testing for successful syncing</td>
<td>The test will be conducted under basic operating procedure of the sensor</td>
<td>The control unit should receive the sync signal from the display</td>
</tr>
</tbody>
</table>

#### 6.2.2.4 IR LED Communication Test Procedure

Below is the table giving a descriptive step-by-step test procedure for testing our IR LED Communication. The test procedure is based off of the test plan we have written in section 6.2.1.4.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Conditions</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The IR LED will be tested for basic functionality</td>
<td>The IR LED will be tested under normal operating voltage of 3 V</td>
<td>The IR LED should illuminate when power is applied to it</td>
</tr>
<tr>
<td>2</td>
<td>The IR Camera will be tested for basic functionality with the IR LED</td>
<td>The IR LED and the IR Camera will be put under normal operating voltages of 3 and 5 V respectively</td>
<td>The Camera should be able to locate the IR light coming from the LED</td>
</tr>
<tr>
<td>3</td>
<td>The Camera and LED system will</td>
<td>The IR LED and</td>
<td>The Camera</td>
</tr>
</tbody>
</table>
be tested for correct communication. A signal will be sent through the IR LED and the camera system will need to pick up the signal and show that it read it in correctly.

the IR Camera will be put under normal operating voltages of 3 and 5 V respectively and system will correctly read in the signal sent out by the IR LED.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Condition</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Test that the lenses are functional by applying a 10 V power source to them and dissipating the voltage with a ground.</td>
<td>Normal operating voltage of 10V</td>
<td>Lense will become opaque and revert back to the clear state when it's grounded</td>
</tr>
<tr>
<td>2</td>
<td>Test the ability of the lenses to shutter at frequencies of up to 120 or 144 Hz by applying a controlled AC signal.</td>
<td>Normal max voltage of 10 V at a frequency of 120 Hz</td>
<td>The Lenses will shutter individually at target frequencies</td>
</tr>
<tr>
<td>3</td>
<td>Test full functionality of lenses when matched to display sync signal.</td>
<td>The test will be conducted under normal operating procedure</td>
<td>The lense should shutter at the times dictated by the sync signal</td>
</tr>
</tbody>
</table>

**Table 12: IR test procedure**

**6.2.2.5 Liquid Crystal Lense Operation Test Procedure**

Below is the table giving a descriptive step-by-step test procedure for testing our Liquid Crystal Lense Operation. The test procedure is based off of the test plan we have written in section 6.2.1.5.

**Table 13: LC lenses test procedure**

**6.2.3 Breadboard Testing**

Testing each of our individual systems was to be done either through direct wiring or through the use of a breadboard. Using the breadboard was also
necessary for prototyping and testing different sections of our circuit. Due to the fact that majority of our parts are SMD parts, it made testing with the breadboard extremely difficult. So instead of utilizing the breadboard we tested the individual parts on an existing board for a pair of 3D glasses while also testing some parts of our circuit for correctness on multisim. Simulation testing was also especially challenging because of the lack of available parts in the multisim software for us to insert into our simulations. The main portions we were able to test out of our testing procedure were the functionality of the IR LEDs, the functionality of the Liquid Crystal Lenses, and the life of the battery.

![IR LEDs](image)

**Figure 21:** Testing IR LEDs for functionality

In the following pictures you can see some of the results of the testing we did with the LCLs. Under no voltage change the lense remains transparent however when we applied power from a 9V battery to the lense, hey immediately became opaque and under reverse voltage became clear again. All in all, both our LED and LCL systems worked correctly.
Figure 22: Comparison of lense under 9 V source (left) and no power (right)
7. Administrative Content

Imperative to executing the project in an efficient, budgeted, and organized manner was an administrative process. This included dividing up tasks, defining and adhering to a schedule, and managing a budget. We now further detail this process.

7.1 Task Division

Given our previous experiences, we divided up the work into several subsystems and tasks. We then assigned each member the most appropriate tasks based on the need and their skillset.

Dane Bouchie, Cp. E.
- Head Tracking System
  - Algorithm
  - Sensor Integration
  - Receive / Transmission
- Interoperability
- System Integration

Matthew Hosken, Cp. E.
- Entertainment Application
- Design Application
- Multi-User Application

Colburn Schacht, E. E.
- Head tracking transmit board
  - Schematic
  - Layout
  - Print
- Head tracking receive board
  - Schematic
  - Layout
  - Print

Eric Smithson, Cp. E.
- Head Tracking Driver (To render system)
- 3D Systems Integration
- Application Interoperability
7.2 Project Milestones

<table>
<thead>
<tr>
<th>#</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>June 2</td>
<td>Divide and Conquer - 10 pages</td>
</tr>
<tr>
<td>2</td>
<td>Jun 9</td>
<td>Updated Divide and Conquer - 10 pages</td>
</tr>
<tr>
<td>3</td>
<td>Jul 7</td>
<td>Draft Senior Design I Documentation - 60 pages</td>
</tr>
<tr>
<td>4</td>
<td>August 1</td>
<td>Final Paper Due - 120 pages</td>
</tr>
<tr>
<td>5</td>
<td>August 1</td>
<td>Have design for project fully completed</td>
</tr>
<tr>
<td>6</td>
<td>August 15</td>
<td>Begin Second Semester</td>
</tr>
<tr>
<td>7</td>
<td>August 15</td>
<td>Begin Hardware Assembly</td>
</tr>
<tr>
<td>8</td>
<td>August 15</td>
<td>Begin Software Development</td>
</tr>
<tr>
<td>9</td>
<td>August 15</td>
<td>Begin Machining and Assembling Housing for Projectors/Screens</td>
</tr>
<tr>
<td>10</td>
<td>September 15</td>
<td>Complete Assembly of 7, 8, 9</td>
</tr>
<tr>
<td>11</td>
<td>September 21</td>
<td>Complete Unit Testing of 7, 8, 9</td>
</tr>
<tr>
<td>12</td>
<td>September 25th</td>
<td>Integration First Pass</td>
</tr>
<tr>
<td>13</td>
<td>September 31st</td>
<td>Integration Second Pass (If necessary)</td>
</tr>
<tr>
<td>14</td>
<td>October 7th</td>
<td>Integration Third Pass (If necessary)</td>
</tr>
<tr>
<td>15</td>
<td>October 15th</td>
<td>Integration Testing (end to end testing)</td>
</tr>
<tr>
<td>16</td>
<td>November 1st</td>
<td>Project Completion</td>
</tr>
<tr>
<td>14</td>
<td>TBA</td>
<td>Committee Presentation</td>
</tr>
<tr>
<td>15</td>
<td>TBA</td>
<td>Senior Design Expo</td>
</tr>
</tbody>
</table>

Table 14: Project Milestones

7.3 Budget and Financing

Our project will have a larger budget compared to most senior design projects due to higher costs in the display technology. The project was still be
self-funded however. Since we are not constrained to a specified amount (as with marketing), the only constraints are personal finance. Between our team, we agreed to limit the project at $3000.

<table>
<thead>
<tr>
<th>Item</th>
<th>Expected Quantity</th>
<th>Budget Allocation Per Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangulation System Electronics (Parts)</td>
<td>3</td>
<td>$100</td>
</tr>
<tr>
<td>Triangulation System Board</td>
<td>3-6</td>
<td>$80</td>
</tr>
<tr>
<td>Desktop System (Rendering)</td>
<td>1</td>
<td>$0 (Pre-owned)</td>
</tr>
<tr>
<td>Integration Software</td>
<td>1</td>
<td>$60</td>
</tr>
<tr>
<td>Base Materials</td>
<td>N/A (1)</td>
<td>$300</td>
</tr>
<tr>
<td>Single Display Face (i.e. Projector/LCD Panel)</td>
<td>3-4</td>
<td>$600</td>
</tr>
<tr>
<td>3D Glasses</td>
<td>3</td>
<td>$100</td>
</tr>
<tr>
<td><strong>Budget Total</strong></td>
<td>N/A</td>
<td><strong>$3000</strong></td>
</tr>
</tbody>
</table>

*Table 15: Project Budget*
8. Conclusion

In this section, we give our final regards to the project by addressing marketability, future improvements to the design, and address our concluding thoughts.

8.1 Improvements

Although our design was effective at our targeted goals, improvements can always be made on any system. We have a few suggestions from our own design that can be improved upon. Most of these choices were not sought in our current design due to availability, financial and time resources. Most improvement focus on the head tracking system system and the system portability.

Although the head-tracking system we designed was successful, it is by no means perfect. The system is reliant upon a low infrared noise. Environments where large amounts of light sources can affect IR communication and constitute low quality, or laggy head tracking. A more robust system could be designed in this area.

The current system centralizes around the use of glasses to create the 3D effect. This is not optimal to the user who, in general, prefers a glasses-free environment for comfort and easy use. A glasses-free system could, in theory, be designed to perform the same effect. Glasses-free stereoscopic LCD displays could be used providing that the viewing angles are large enough for head-movement during use of the system. If not, the system could be mechanically adjusted to accommodate for small viewing angles. For the head-tracking effect, advancements in vision processing could be used to track a user’s head position. This is a more-difficult task to do at continuous high speeds and minimal computation time for desired latency bounds. In addition, to even find accurate orientation. With enough time and resources, this can probably be achieved.

Portability improvements could also be made, as mentioned in the previous section. Our system was designed as both a research base and a presentation. Portability was unnecessary given the mechanical engineering needed, therefore improvements could be made. For compactness, LCD screens could be used (whenever a polarized, or viewing angle system). An embedded system could be used for the rendering system with the space created inside the display (since the LCD screens do not require a hollow display space). The system could support a large rechargeable power system at the base of the display for mobility. The cameras or other tracking system could then be build
into the display themselves. This would render the system much more mobile than the existing state.

8.2 Marketability

As discussed, our product was never designed around marketability. Removing this constraint from concern gave us freedom to develop a high quality proof-of-concept that future designs could be based off of. Nevertheless, we follow-up by addressing the concerns around marketability of our project.

It’s trivial that a marketable product needs a market. This was the first concern if our product was to be marketed. In the industry, most applications have pushed towards VR and AR. While our device provides a unique experience, it lacks features that may have brought it to a large audience. AR applications are designed to be portable so they can be used in many different locations, applications, and purposes. Our system is currently heavily constrained to fixed location. Before this device would ever be marketed, it would need to be much more mobile. Current VR applications are not as versatile in this manner as AR is. However, VR offers a different experience where location doesn’t matter, as you are fully placed in a virtual environment. It is for this reason, our device would need to be made more compact.

It is also hard to gauge where our product stands when compared to AR or VR. While each system is diverse enough to give its own experience, VR or AR specifically might become more popularized.

In addition to that, the overall cost of the device is very high. This is mainly due to the 3D DLP technology needed to display. As it stands now, any consumer-level product would be far too expensive. However, several improvements could be made to the overall cost on a large scale. The first is using lower-cost DLP technologies while integrating the DLP system directly. While this may reduce the cost by several factors, overall the cost would still be relatively high. For DLP technology to be sought in a consumer-level implementation, it would require a substantial drop in price of DLP technology. A more reasonable solution to a lower cost device is to custom create 3D displays in the display shape. Given that our implementation was flat surfaces, it is reasonable to think it would not be terribly hard to adapt to an LCD system. On a mass-market scale, this might be enough to reduce the cost to a reasonable value. However, this requires the technology and market size to manufacture such displays at a low cost.

For all these reasons, an initial market for the device is not too practical.
8.3 Concluding Thoughts

We successfully researched, designed and tested a 3D holographic device. Using the design process, we first put effort into study of technology. We then applied this to part selection. After architecting each system, both software and hardware design were finished. Finally, each component and subsystem were tested so that on a full scale, can be implemented.

This process has given our initial goal of applying learnt skill to academic purposes. We then hope the full-scale version emanates our interest and skill in computer engineering and electrical engineering.
9. Implementation

In this section, we go over the final implementation, testing of the system, and the realization of the the design.

8.1 HeadTracking

The head-tracking system was developed using the design. Not all aspects of the design were realized which we will cover in detail in the final subsection. Overall, the individual pieces were developed on their own. And the important baseline of the system was fully integrated.

Unit testing using the C++ Google Test Framework was used for particular cases. Real video of the infrared tracking was recorded in several scenarios. The particular cases were tracking a single LED with minimal movement, tracking a single LED with rapid movement, tracking multiple LEDs together with direct perspective, and tracking multiple LEDs with difficult (side) perspective. The later case was needed to test the extent in which multiple LEDs blended into each other. In addition, a calibration test image was used to test the calibration system using fake coordinates. Communication unit tests verified that the device adhered to the serial protocol we defined in the design so that integration with the DLL (covered further in the next section) took minimal time. With all of these tests in place and testing, including the application tests covered in the next section, the final tracking testing could be performed.

Integration involved detecting the real-world LEDs and system itself. First the program was updated so that the exact coordinates of the LEDs relative to the center of the glasses were known. Once updated, we first tested the ability for the system to track the LEDs themselves. After seeing correct perspective changes based on the LEDS in projection to the camera, we concluded that the base-line IR tracking was successful.

The stereoscopic effect was tested using a particular system, but unfortunately, added extra time need when ported over to the final system. Because of this, it was not fully utilized. The bluetooth module was tested during prototyping, and was able to transmit data, and the IMU sensor was tested on its own using an arduino. Unfortunately due to issues with shipping and the time constraints, the parts were not available in time for the system to see the accelerometer fully contributing to the head-tracking precision.
8.2 Interfacing

We were able to successfully integrate hardware interfacing to our application engine. The pi used OpenCV to track the position of the four LEDs as they were presented on the head of the user. The Raspberry Pi was able to successfully communicate these coordinates using the command schema we developed for this purpose. We initially tested our command schema by sending over 16 bytes of data and waiting for the pi to send it back. Initially we struggled with our software design and getting the right amount of bits to transfer wasn’t an easy task. We initially mistook where the software was sending the data and we ended up receiving it out of order by creating two threads, but we were able to sort it out. The program successfully sent over the correct bytes in order, and were interpreted via COBS correctly. The command schema is listed below:

1. ResetCalibration
   a. Type: 0x01
   b. Data: byte CameraNum (0x00 = primary, 0x01 secondary)
2. AddCalibration
   a. Type: 0x02
   b. Data: byte CameraNum (0x00 = primary, 0x01 secondary)
   c. Data: float x,y,z camera position
3. UpdateEyeDistance:
   a. Type: 0x03
   b. Data: float EyeDistance
4. ConsoleOutput:
   a. Type: 0x04
   b. Data (X): string value
5. Update Eye Position:
   a. Type: 0x05
   b. Left Eye
      i. Float x, y, z (float.NaN if not detected)
   c. Right Eye
      i. Float x, y,z

8.3 Application Engine

The application engine was built using Unity and a C# DLL. The DLL was created as specified in the design, and worked successfully, passing the unit tests created.

The application engine was then attached to the DLL. To integrate it, the DLL was compiled and then placed directly into the engine. A small modification was
made so that the DLL had a singleton instance, and could be easily accessed through the engine without re-instantiating. This was then hooked up to the configuration handler and the tracking handler. The configuration handler was used to edit particular calibration settings on the head tracking system, and also start the calibration process itself. The tracking handler was connected to the tracking position event from the DLL and updated the view and perspective based on the updated eye positions. With this in place, testing was performed by re-compiling the DLL with fake events, and debug output so that all handlers could be correctly utilized. Once finished the DLL-Application engine integration could be completed.

When attached to the head tracking hardware (the Raspberry Pi). We first verified the communication functionality by using debug statements. We then thoroughly tested each of the commands that were implemented in both directions by testing debug values. In is important to note this took place after the head-tracking communication unit tests were tested as well, which sped up the integration process. When all debug output was tested, we then tested a sample case where the tracking performed updates in a circular track. With this, we were able to see output on the application engine where the camera was updated as expected (in a circular track). Throughout this testing we concluded the Application Engine - DLL - Tracking System successful.

![Figure 23: Application during integration tests](image_url)
8.4 Hardware

Upon final integration of all the separate systems hardware wise, we continually ran into issues with the IMU system. On dev board and unit testing we could get the IMU to read out acceleration and gyroscope data but could not get the chip to transfer this same data when integrated with the bluetooth unit on the final board. All of the correct areas were delivering power where they needed to be however there were no results to show for it. We even tried switching the chip to see if it was faulty but just as before no data was reaching the display unit for tracking. An error may just be caused by the way it is laid out and in the future we would try to relayout the board to ensure a working unit.

Outside of the IMU unit we managed to get the rest of the hardware working and calibrated so the display unit could track the location of the glasses and make the necessary changes on the screen to ensure player immersion. Some features we would like to add in the future to further enhance the playing experience for the user hardware wise are increased accuracy from a larger number of IR “targets” for the camera to take into account, a smaller more compact unit overall that does not interfere with the user’s’ viewing experience, and a rechargeable battery option so users are not required to replace batteries for the unit.

First off, implementing more IR targets on the glasses may be difficult algorithmically for the software portion of the project, however they would largely enhance the 3D experience for the user. Adding these extra LEDs would not be difficult on the hardware side and would only take up a little more space wherever you plan on putting them. Having all these extra points of reference for the camera would make it more able to accurately track the wearer’s head movement thus reducing the necessity of the IMU for high accuracy head tracking.

Creating a more compact, interchangeable unit while also replacing the need for replaceable batteries I believe would really set this unit over the moon. As of now the unit is 30x30x40mm, however by using smaller components and better 3D printing materials we could shrink it and form it into a more sleek design while also making room for a more user friendly rechargeable battery. While we’re at it we could also look to improve the mounting system of the casing so it becomes universal for all 3D glasses, all you would need to do is clip it on. With these minor changes I believe this unit could have large promise.
8.5 Improvements

It’s important to note the final status of the project, which did not implement the full design, but the baseline. Due to issues with shipping and time constraints, not all parts were able to be implemented. In particular, the Bluetooth system with the IMU sensor was not able to be implemented in the system, which would have increased the precision in the design. However, the system still passed the overall requirements, and is ready for further improvements in future research. The housing itself is still in a developmental stage which would
have added to the completeness of the project, but also was not necessary to the fundamental design. Finally, the stereoscopy was tested, but only on a individual system. Because of this, it was not able to be used across all of our systems, and still requires further integration.

All of which would assist in improving the system greatly. The first and biggest improvement would be the addition of the 3D calibration. The 3D calibration makes up for the lack of known camera locations, hence the lack of a housing. With it fully implemented, full 3D perspective would be correct and the system would render a much more accurate projection than the final implementation. All of which is described in the design. Currently the system only tracks in reference to the camera, and not the center of the housing (or world coordinates) because of it.

Following, the housing would greatly benefit the system by adding not only a unique display, but also, if precise enough, remove the need for calibration. Overall, this might improvement the accuracy in addition to the user experience. In addition, allow multiple systems to be setup so users can collaborate during use in the application.

Once the 3D calibration was further in place, the bluetooth will be utilized to fuse the lower frequency data with the higher frequency data for more responsive and accurate movement. With this data, the kalman filter will properly fuse data to produce the improved results.

Similarly, an improvement can be made by using multiple cameras in the system. The final implementation only saw one camera in use. More cameras can be added (with different perspective cameras can be added with different perspectives to have a larger measurement data set. The algorithms to do so, however, need further research, but are in existence. The housing also provides a better way to handle this to avoid calibration every camera in the system. It's still important to note that the cameras still require a one-time "simpler" calibration to calibration the lens distortion (discussed in detail in Section 5).

**8.6 Final Thoughts**

We saw our system go from design, to prototyping, testing, and integration. On an individual level, each component was tested and passing, using unit testing and integration tests. Overall, we were successfully able to to build the baseline of our system, and developed a research platform for later improvement and use.

As discussed above there were many improvements in the implementation that could be followed from the design, due to the ambition of the project. Although, the requirements initially set were fully realized.
With this, we see the success and potential in the current system and design. Throughout the process, we have learned the challenges, and constraints in developing an ambitious project, have developed and practiced our engineering skills, and saw the requirements completed with the given constraints.
Appendix A - References


Appendix B - Copyright Permissions

OpenCV (Including Documentation)

By downloading, copying, installing or using the software you agree to this license.
If you do not agree to this license, do not download, install, copy or use the software.

License Agreement
For Open Source Computer Vision Library
(3-clause BSD License)

Copyright (C) 2000-2016, Intel Corporation, all rights reserved.
Copyright (C) 2009-2011, Willow Garage Inc., all rights reserved.
Copyright (C) 2009-2016, NVIDIA Corporation, all rights reserved.
Copyright (C) 2010-2013, Advanced Micro Devices, Inc., all rights reserved.
Copyright (C) 2015-2016, OpenCV Foundation, all rights reserved.
Copyright (C) 2015-2016, Itseez Inc., all rights reserved.
Third party copyrights are property of their respective owners.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice,
  this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

* Neither the names of the copyright holders nor the names of the contributors
  may be used to endorse or promote products derived from this software
  without specific prior written permission.

This software is provided by the copyright holders and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed.
In no event shall copyright holders or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages
(including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.
Appendix C - Software code

C# DLL Code

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using static System.IO.TextWriter;
using System.Threading.Tasks;
using System.Threading;
using System.IO.Ports;
using UnityEngine;

namespace ClassLibrary1
{
    public class MyUtilities
    {
        SerialPort com;
        byte[] buffer;
        Thread thread;

        public struct data
        {
            public int gyrox;
            public int gyroz;
            public int gyroy;

            public int accelx;
            public int accely;
            public int accelz;
        }

        public data GetData()
        {
            data stuff = new data()
            {
                // This is the same order I expect to receive the
                bytes in
                gyrox = buffer[0],
                gyroy = buffer[1],
                gyroz = buffer[2],

                accelx = buffer[3],
                accely = buffer[4],
                accelz = buffer[5],
            };
            return stuff;
        }
    }
}
public void SetData()
{
    while (true)
    {
        byte[] temp = new byte[6];
        com.Read(temp, 0, 6);
        for (int i = 0; i < 6; i++) {
            buffer[i] = temp[i];
        }
    }
}

public MyUtilities()
{
    com = new SerialPort();
    com.PortName = "COM4";
    com.BaudRate = 19200;
    com.DataBits = 8;
    com.Parity = Parity.None;
    com.Open();

    buffer = new byte[6];

    // thread = new Thread(new ThreadStart(SetData));
    // thread.Start();
}