# VR HAND SIMULATION

By

Alex Brannick

Daryl Drake

Final Report for ECE 445, Senior Design, Spring 2019

TA: Dongwei Shi

1st May, 2019

Project No. 67

## Abstract

This report details the design and implementation of the device known as the VR hand simulator. It was designed to created a more immersive VR experience, and consists of both a headset and bracelet module.

The headset uses a webcam to capture images of your hand. This is used in a real time hand modeling pipeline to create 3D hand data. From here, the data is sent from a server over LAN to the Oculus Go, where it is populated in Virtual Reality.

The haptic feedback bracelets consist of a small micro controller that receives IR codes to vibrate a small motor. It receives these codes from the server, and it is triggered when your 3D hand interacts with an object in VR.

These modules came together to create a simple demo game that recreates your hand in realtime, and these findings show a promising future for hand modeling technology.

# Contents

# 1. Introduction

## 1.1 Objective

Ever since Oculus VR was acquired for $2 billion by Facebook [1], there has been a shift by larger companies to start developing and investing in VR technology. Companies such as HTC and Sony created their own headsets to name a few, and the increase in competition lead to a huge push for VR. In particular, these virtual reality headsets had a large presence in the gaming industry, leading to many game developers creating new exciting games for these platforms. While the quantity and quality of these games has been steadily increasing, the VR headsets themselves have not changed much in the past decade. These games are meant to provide an immersive experience for the user, but the platforms they are made for have been lacking in innovation. The LA times wrote an article in 2018 stating that funding for VR startups had fallen 46% since 2017 [2]. This lack of funding was due to concerns by investors that the VR market itself had not been able to reach a large enough number of consumers to become as lucrative as expected.

In order to help revive the VR industry, the goal of the VR hand simulator is to create a VR headset attachment that creates a more captivating experience for the user. With this device, it will give the ability for users to perform complex interactions with virtual reality using just their hands as the controllers. Along with hand modeling, the simulator will use haptic-feedback bracelets to give gamers a sense of touch when interacting with objects in the game. Furthermore, the device will include a Unity API or plug-in for any game developer to easily interface with the project. By including these parts in the design, we believe the VR hand simulator will be able to stimulate the interest in VR technology once more.

## 1.2 System Overview

The first portion of the VR hand simulator is the headset, as seen in figure 1. This portion of the project includes the RGB webcam, which will be taking in frame data at a rate of 30fps. The frames are passed to the hand modeling pipeline, where the hand inferencing is done to gather 3D joint data of the users hands. This data is passed to the python server where it is sent over LAN to a client script within the Unity Application. The data is finally populated within the Oculus Go, and can be seen on the screen at this point. When the hand touches an object in the game, an event is triggered, sending a code to vibrate the bracelet from the client back to the server. This code is passed over a serial connection to the headsets IR Transmitter, which sends it out as a TV remote signal to the bracelet.

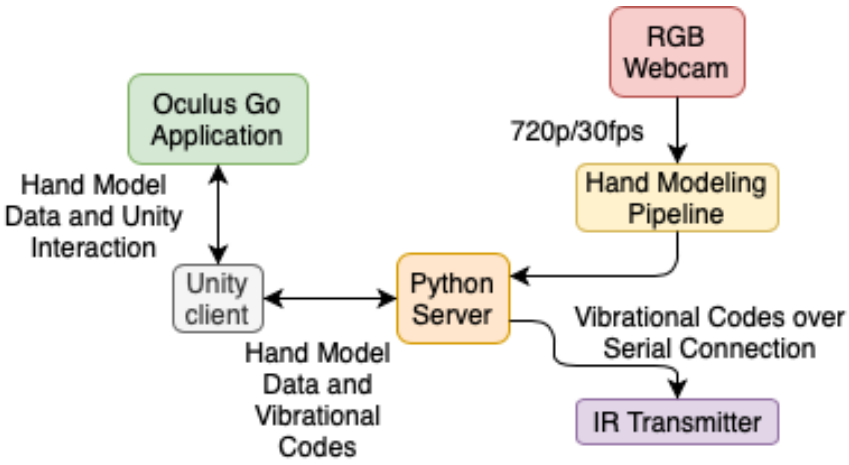


*Figure 1*. Block Diagram for the headset portion of the project. The diagram shows the direction in which data is being passed between modules, and what information is being sent. The Oculus GO Application and Unity client are implemented on the Oculus Go itself, while the rest of the components are ran locally on a laptop.

Seen in figure 2, the second portion of the system is the haptic-feedback bracelet. IR remote signal codes from the IR transmitter are received as raw IR data by the IR receiver. This data is passed to the Adafruit trinket m0, a micro controller that does the bulk of the work for the bracelet. At this point the trinket decodes this raw IR data and determines if the received code is meant to vibrate the motor on the bracelet. If a proper code was detected, then the trinket varies the output from its GPIO pin to change the voltage across the vibrational motor using a pwm circuit. Finally, the LED is flashed to signify that a code was successfully received and executed.

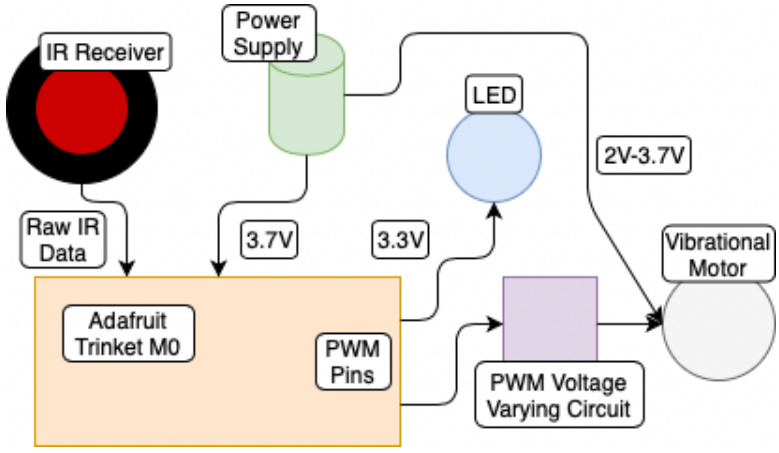


*Figure 2*. Block Diagram for the bracelet portion of the device. The block diagram is meant to show the various components of the bracelet as well as how they are powered. This entire circuit fits within the size of someones wrist, and provides three different levels of haptic-feedback to the user.

## 2 Design

Our design changed quite a bit over the course of the semester, but we were still able to complete the prototype. These are the updated components of the project that made it into our final design, with engineering explanations as to why new components were substituted in for others.

### 2.1 RGB Webcam

In order to get frame data for our pipeline, we needed an RGB camera to mount to the Oculus Go headset. We chose the Logitech C270 webcam for its relatively cheap price, 720p resolution, and frame rate of up to 30fps [3]. The camera worked directly out of the box without the need to install any drivers, but we ran into an issue where the image was undersaturated and very dark. In order to change this, we used pythons openCV library to tune the webcams parameters to get a more quality image.

### 2.2 Hand Modeling Pipeline

The part of the project that generated the 3D hand data was the hand modeling pipeline. Using a set of frameworks and closed-source binaries, a group of researchers managed to create a pipeline that imposed hand models on people from video files [4]. Taking inspiration from their work, we modified their pipeline scene in figure 3 to work with a webcam in real time that can generate 3D hand key points.

This pipeline consists of three main parts, with the first being a bounding box predictor made specifically for identifying hands. The predictor uses a model architecture known as YOLO, or you only look once. It has been shown to be one of the fastest and most accurate networks with speeds of up to 30fps and mAP 0f 57.9% [5]. It takes in the frames from our webcam and outputs the bounding box location for the hands in the image.

From here the bounding box for the hand is cropped and passed to OpenPose, a framework for real-time body pose estimation [6]. Using the image of the hand, OpenPose runs a CNN to determine the peaks in the frame. The framework then uses these peaks to estimate where the 21 hand joints are located in the image. It outputs these 21 key points as 2D coordinates in the frame.

Finally, the key points are passed to the Inverse Kinematics model to be estimated in 3D. The Inverse Kinematics model was created by a third party, and has been made closed source. Luckily, they provide closed source binaries for use with python on an ubuntu os. Passing in the 21 hand key points, the model can use this information to estimate these same key points in 3D. What is left are depth estimates for each joint, giving us 3D hand data for populating are game with.

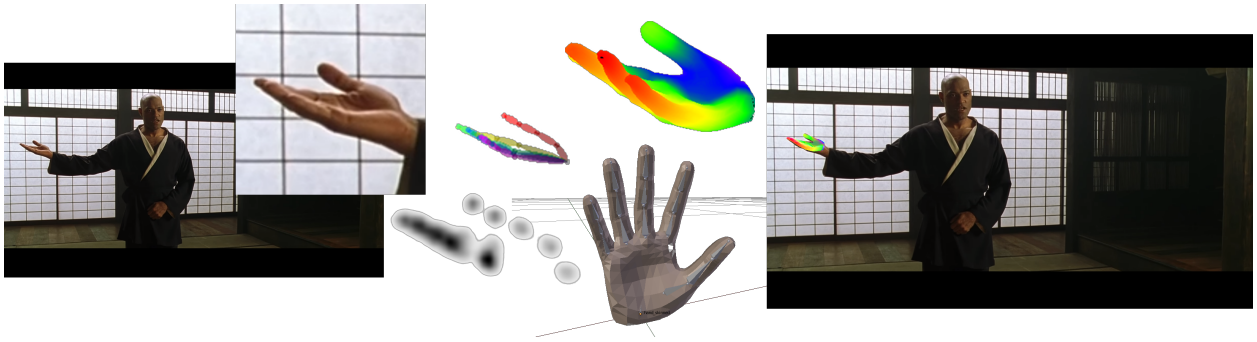

*Figure 3*. Here is an image from the research paper that helps visualize the pipeline. The image to the left represents a frame in the video. The hand is identified and cropped by the YOLO model. It is passed to OpenPose next, where these peaks are turned into 2D joint estimations. Finally, these 2D points are converted to 3D. The image to the right shows these hands applied to the original frame from the video.

## 2.3 Unity Interface/Plugin

In order for our device to be widely available to developers, we had to make some type of unity interface or plugin for creating games with it. After coming to the conclusion we needed a network architecture to get the data on the Oculus, we created a Unity client for this component. The unity client polls the server every 50ms, or 20 times a second. It receives the 3D hand key points from the server after each request and allows a developer to populate their game with this data.

While the client is constantly polling the server for this data, it is also able to send messages on command back to the server. If an event is triggered in the game that requires the bracelet to vibrate, the client can send a message back to the server to trigger the bracelet. All of this data is sent over the LAN, which is the quickest way to send data to the Oculus without having access to the usb port.

## 2.4 Python Server

The python server is used to interface the pipeline and bracelet with the Oculus Go itself. It is an echo server, so any time a message is received from the Unity client it can send a message back. When the client polls for hand data, the server runs one iteration of the pipeline and sends the results over the network back to the game. When a message is received to vibrate the motor, our server relays this information over a serial connection to the IR transmitter.

In order to send the data over the network, we chose to use pythons secondary network library called asyncio. While pythons socket library is the main library for network programming, we chose to use asyncio because it allowed for asynchronous operation and proved to be much faster for our use case.

## 2.5 IR Transmitter

All codes from the server have to be sent to the IR transmitter in order to be flashed to the bracelets. When the python server receives a code from the Unity client, it is sent over a serial connection to the micro controller for the IR transmitter. We use PySerial, a python library for serial data transfer. The micro controller reads in this data at a baud rate of 9600 and formats the data for the transmitter. Finally, an IR library for Arduino was used in order to send the vibrational codes from the micro controller. The library encodes the data into a series of 1s and 0s that the IR transmitting diode will flash to the controller.

## 2.6 Adafruit Trinket M0

We chose to use the Adafruit Trinket M0 due to its extremely small size and ease of use. In figure 4, it is shown that the device is smaller than a quarter. The micro controller uses Circuit Python, a derivative of python meant for controlling devices like the trinket. While the small size was needed to fit the device on our wrist, it did not give much space on the chip for our source code. We had 4mb of memory to store the code, so we deleted unnecessary libraries and added compiled .pyc binaries for any libraries that were required.

The trinket used a library for decoding the IR data, and this allowed us to review any signals being sent to the controller. From here, we compared the received signal to saved codes to determine if the trinket should vibrate the motor. At this point, we varied the duty cycle on the GPIO pin connected to the PWM circuit in order to vibrate the motor.

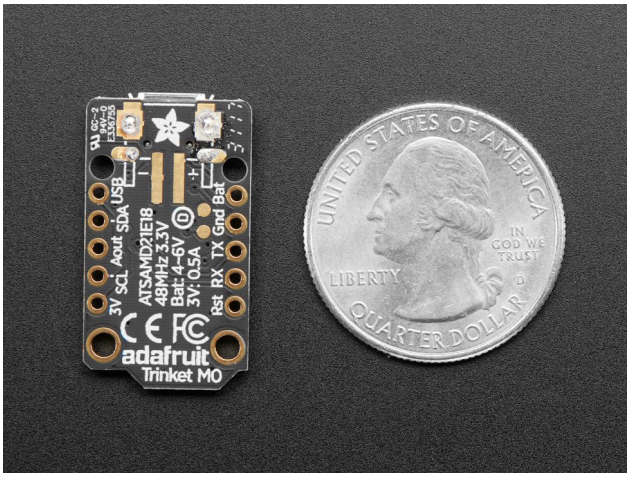


*Figure 4*. The trinket was chosen for its small size. Here is an image of the Adafruit Trinket M0 when compared to a quarter.

## 2.7 PWM Circuit/LED

In order to vibrate the motor, we used a pwm circuit to vary the voltage across the motor. Our original plan was to connect the motor directly to the GPIO pin on the trinket, but the pin could only supply ~20mA of current when the motor needed at least 80mA. Due to the relatively low current that could be drawn from the pin, we went with a pwm circuit where the GPIO pin would control the voltage to the motor. It consisted of a single npn transistor, a resistor, and a diode that was in parallel with the motor. By varying the duty cycle from the GPIO pin, we were able to change the voltage being supplied to the motor. This allowed for varying levels of vibrating intensity in our bracelet. As you can see in figure 5, the GPIO pin controlled the transistor instead of directly changing the motors speed.

The LED added to our circuit was meant mainly for debugging and feedback. We used it to signify when a code was received by the bracelet, and it also gives the user visual feedback of when a code has been received.

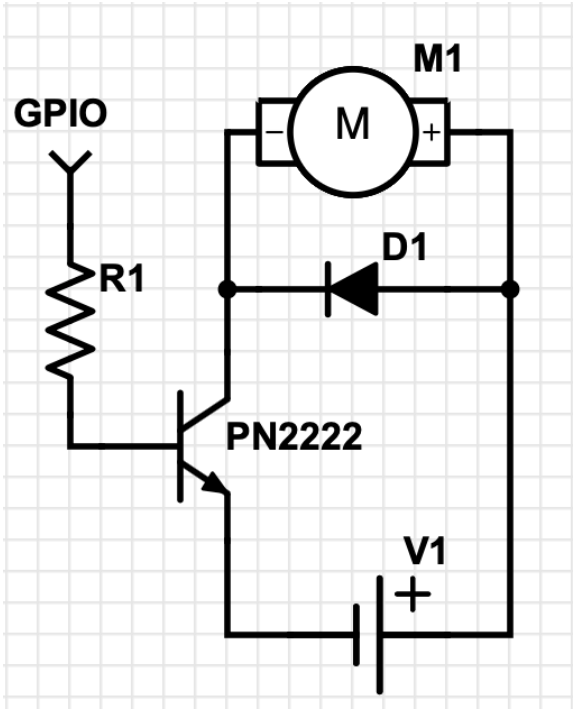


*Figure 5. Here is the schematic for the PWM circuit. The GPIO is used to control the transistor, which varies the voltage to the motor.*

### 2.8 IR Receiver

The IR receiver is another major component to the bracelet, and serves as the point where all data is supplied to the device. IR codes are sent from the headset to the bracelet, and this data is received and passed to the micro controller. When infrared light from the transmitter is seen by the receiver, it is flashed to a GPIO pin on the micro controller as a raw, unformatted signal. The receiver does not manipulate this information at all, and leaves this job to the trinket for the signal to be decoded.

### 2.9 Power Supply

Since the bracelet is meant to be worn on the wrist, we went with using a portable battery to power the circuit. After some research, we chose a small 3.7V, 350mAh battery from Adafruit Industries due to its small size and longevity. While LiPO batteries are known to have issues [9], we believe that the conditions in which we are using this battery will prevent it from overheating or combusting.

Outside of concerns about the battery itself, we went ahead and calculated the expected battery life when it is connected to the bracelet. Using the method detailed from [10], we estimated the total current draw of the trinket to be about 35mA. After running our estimates through the equations seen in table 1, we calculated that the device could last upwards of 2 hours.

**Table 1 Battery Equations**

| | |
|---|---|
| BL = Total current draw (A) * Hours needed (H) | Basic battery life |
| BL* = BL (mAh) / Cycle Life (%) | Battery life with cycle life |
| BL** = BL* (mAh) / Rate of Discharge (%) | Battery life with rate of discharge |

## 3. Design Verification

During the process of building this prototype, we had to make sure that each component met a baseline of requirements. These requirements were laid out at the start of this project, and they had to be completed and verified in order for the device to be completed. Details of the verification process will be explained here, while the original R&V tables can be found in the appendix below.

### 3.1 RGB Webcam

In order to verify that the camera was working, we hooked up the camera to the laptop. We used the openCV library in python and ran a short script to display the cameras view. The cameras original settings caused each frame to be undersaturated and dark, so it required us to increase the exposure time of the camera. We use openCV to grab the frames in the hand modeling pipeline, so this verification guaranteed that the pipeline would receive quality frames.

### 3.2 Hand Modeling Pipeline

The hand modeling pipeline was one of the more complex components, but the testing was relatively easy. After verifying the webcam was providing frames at 30fps, we needed to show that the pipeline is outputting the 3D hand key points properly. The way we did this was by displaying these points in a python particle viewer. Holding our hand steady, we paused the frame and viewed the key points from a 3D environment. Once we verified that these points resembled are hand, we began testing the speed of the pipeline. By adding a timer to the algorithm, we calculated that the pipeline could reach speeds of up to ~12fps and averaged about 10fps.

### 3.3 Unity Interface/Plugin

In order to test the unity interface, it required verifying that the hand model could be received and the vibrational codes were being sent. This was the last part of the project, and was the final step in verifying it worked. We created a simple demo scene in unity to test that the process of sending and receiving data was working. By pressing a button on the controller, it sent a message through the unity client to vibrate the motor. We verified that three different codes could be sent this way. Conversely, the unity scene took the 3D hand data and populated it to the screen. Viewing the hand recreated in the environment was enough verification that the data was being sent correctly.

### 3.4 Python Server

The python server requires similar verification as the Unity client, except that it needs to be able to receive vibrational codes and send 3D data. Again, we use the same verification process as the unity client. By creating a simple demo, we can verify that all codes and hand data can be sent and received by both ends.

### 3.5 IR Transmitter

The final component of the headset is the IR transmitter. The transmitter needs to send all vibrational codes to the haptic feedback bracelet. In order to verify that it meets the requirements, we needed to show that it could send all three vibrational codes to the bracelet. We created a simple test script to send these three codes in a row, and this was verified by seeing if the bracelet would vibrate. At this point we had finalized the bracelet, so all three vibration codes would lead to the bracelet moving.

### 3.6 Adafruit Trinket M0

Testing the Adafruit Trinket M0 uses all components of the bracelet. It requires the trinket to be able to decode raw IR data as well as pulse the vibrational motor. In order to set this up, we had to download circuit python to the device. There is a library for decoding raw IR data, so we used that to determine if the IR codes were being sent correctly. From here, we just varied the duty cycle of the GPIO pin for the pwm circuit to test different levels of vibration for the motor.

### 3.7 PWM Circuit/LED

The pwm circuit was testing simultaneously with the trinket. After building the circuit described in the design portion of this report, we tried vibrating at different intensity levels. There were three levels of vibration that we tested on each bracelet. This was done by varying the duty cycle from the trinkets GPIO pin.

### 3.8 IR Receiver

The IR receiver was tested directly with the Adafruit Trinket. Since there was no way of telling if the receiver worked until we hooked it up, we decided to test it from the micro controller itself. We found 3 different signals that were sent from my tv remote and tried to see if we could get the codes from the IR receiver. Using the IR decoding library from circuit python, we were able to see the codes displayed on the monitor.

### 3.9 Power Supply

The power supply was one of the less strenuous portions of the project to test. We bought our LiPO batteries from the same manufacturer as our microcontroller, so we knew that it would provide enough power to our circuit. After soldering it to our board, we verified that we were receiving the correct 3.7V from the battery using a voltmeter. Afterwards, we focused on testing the longevity of the bracelet by vibrating the bracelet every 2 seconds until the battery was almost dead. The battery lasted over 4 hours, but it will reduce in charge length over time.

# 4. Cost & Schedule

## 4.1 Parts

| Table 2 Parts Costs | | | | |
|---|---|---|---|---|
| **Part** | **Manufacturer** | **Quantity** | **Retail Cost ($)** | **Bulk Purchase Cost ($)** |
| Oculus Go | Oculus | 1 | 199 | 149 |
| IR LED Diode | Adafruit | 1 | 1.05 | 0.55 |
| RGB Camera | Logitech | 1 | 21.99 | 17.99 |
| Adafruit Trinket | Adafruit | 2 | 8.95 | 7.95 |
| IR Receiver | Adafruit | 2 | 1.95 | 1.45 |
| Vibrating Motor | Adafruit | 2 | 1.95 | 1.45 |
| LiPO Battery | Adafruit | 2 | 6.95 | 5.95 |
| PCB | PCB Way | 2 | 3.10 | 2.10 |
| **Total** | - | - | 267.84 | 205.34 |

## 4.2 Labor

In order to accurately estimate the cost of this project, the cost of labor must be taken into account. This was done by referring to the Average Computer Engineering Salary for a graduate of the ECE department in 2014-2015. [7] We found the average salary to be $84,250, which came out to about $40.50/hour when using a 40 hour work week standard. We estimate that it took us about 10 hours a week to complete the prototype over the course of 10 weeks. With all of this being said, we have calculated the total cost for both of us below:

Total = 2 partners x $40.50/hour x 10 hours/week x 10 weeks = $8100

## 4.3 Schedule

| Table 3 Semester Schedule | | |
|---|---|---|
| **Week** | **Alex** | **Daryl** |
| 2/23/19 | Practice Design Review Presentation | Practice Design Review Presentation |
| 3/2/19 | Finalize PCB schematics for bracelets | Begin software development of bracelets |
| 3/9/19 | Begin software development for rgb cameras, serial data transfer, and IR transmitter | Finalize software development of bracelets<br>Begin setup for pipeline environment |
| 3/16/19 | Improve efficiency of hand model | Improve efficiency of hand model |
| 3/23/19 | Develop second round PCB for project | Begin researching data transfer to Oculus Go |
| 3/30/19 | Finalize hand modeling software<br>Start research on data transfer to Oculus | Finalize hand modeling software<br>Continue working on data transfer for Oculus Go |
| 4/6/19 | Build full prototype of haptic feedback bracelet | Begin design of client/server architecture |
| 4/13/19 | Fully test hand modeling pipeline and haptic feedback bracelet | Complete design of client/ server architecture |
| 4/20/19 | Begin presentation and final report | Create simple Unity demo game<br>Iron out last minute bug fixes and clean up/comment all code |
| 4/27/19 | Practice Presentation, finalize report | Practice Presentation, finalize report |

# 5. Conclusion

## 5.1 Accomplishments

After working on the project for the past two months, we were able to successfully provide a proof of concept for our device. While some aspects of the project had to be modified to realize our original goal, we were able to complete a fully functioning version of our device.

The final product stayed true to our original design for the haptic feedback bracelet, with an IR receiver taking in data and the trinket decoding it. Again, the codes specify three separate intensities that the bracelet can vibrate at. All pieces worked when soldered on to our PCB, and the device was still small enough to fit on a persons wrist.

For the headset, we were able to get the hand modeling pipeline inferencing 3D hand data at a speed of up to 15fps. Along with the pipeline, the IR transmitter was able to successfully send all codes to the bracelet accurately. Unfortunately we had to run our project from the laptop, but we were able to create a simple demo game in Unity to show the hand model on the Oculus Go.

The demo game that we created showcases all data being received by the Oculus as well as data being sent from it. By pressing three separate buttons, they each send vibration codes to the IR transmitter to buzz the bracelet. The hand data is also populated to the screen in real time from the pipeline over our network architecture.

## 5.2 Uncertainties

The final design of our project saw some changes when we began development. In the original design, we planned on transferring all data over a serial connection to the Oculus Go instead of over the network. We also originally planned on running the entire pipeline from a raspberry pi with the Movidius Neural Compute Stick instead of on the computer.

The reason we were unable to use a serial interface for data transfer was due to the closed nature of the Oculus Go platform. All ports were left closed on the device, and there was no way for a user to gain root access to open them. While Unity and Oculus did provide API's for the usb interface, they were only for programming buttons on an Oculus or third party controller. Sending raw data over these API's was not supported.

Outside of the Oculus being a terrible platform to develop with, we also ran into issues when trying to use the pipeline with the raspberry pi. Parts of the pipeline were closed source binaries, so we did not have access to the original code. In order to use the compute stick, it required us to change some portions of the Tensorflow

code to use the NCS's APIs. Since we only had binaries for the last part of the pipeline we were unable to change this part of the code. Furthermore, without the compute stick to increase the inferencing time, the raspberry pi was just too slow to run the pipeline for our project. After coming across these issues, we eventually settled on using the laptop in the final design to finish the project by the due date.

### 5.3 Ethical considerations

During the project, we made sure to closely adhere to the IEEE code of ethics described on their website. Within their code of ethics, rule 9 states that we should "avoid injuring others, their property, reputation, or employment by false or malicious action." [8] Our design is not its own entity, but rather an attachment to an existing product. As a third party device, we need to make sure that the VR hand simulator is branded that way and has no affiliation with the major VR companies. If our product were to affect their reputation in a negative way, that would be a direct violation of this rule. In order to prevent this, we will work on explicitly stating on our packaging and the device that our product is not made by these VR companies. Furthermore, if the product began gaining popularity we would need to work with these VR platforms to make sure our device continues to receive their approval and support.

### 5.4 Future work

After finishing the first iteration of our device, we believe there is still a lot of room for improvement. One of the first ideas to improve the design would be to expand the haptic feedback bracelet into an entire glove. We currently support three vibrational intensities from a single motor on each bracelet. By adding more motors for each finger or even each joint, it would create a more realistic sensory experience for the user. Adding more motors to our design would be relatively easy as well, but we may have to upgrade to a larger power source and micro controller in order to power all of them.

Along with a haptic feedback glove, the client/server architecture we created for transferring the hand data to the Oculus could be improved dramatically. The bottleneck of our design is not the hand modeling pipeline itself, but the transfer of data over the network. We believe that we can reduce this latency by as much as 50% through improving the architecture, but in order to drastically improve the speed we would need to redesign this portion of the device. We were unable to get the data sent over a serial connection due to lack of documentation and time, so with more time to research this subject we may be able to see results.

## References


[1] Virtual reality. (2019, February 06). Retrieved from https://en.wikipedia.org/wiki/Virtual_reality

[2] VR gets reality check with significant decline in investment. (2019, January 13). Retrieved from https://www.latimes.com/business/hollywood/la-fi-ct-virtual-reality-investment20190113-story.html

[3] Logitech HD Webcam C270. (2019, May 1st). Retrieved from https://www.bestbuy.com/site/logitech-hd-webcam-c270-black/9928354.p?skuId=9928354

[4] P. Panteleris, I. Oikonomidis, and A. Argyros, "Using a single RGB frame for real time 3D hand pose estimation in the wild." [Online]. Available: http://users.ics.forth.gr/~argyros/mypapers/2018_03_WACV_rgbmonohand.pdf. [Accessed: 01-May-2019].

[5] J. Redmon, "YOLO: Real-Time Object Detection," *YOLO: Real-Time Object Detection*. [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed: 02-May-2019].

[6] V. Gupta, "Home," *Learn OpenCV*, 27-Feb-2019. [Online]. Available: https://www.learnopencv.com/tag/openpose/. [Accessed: 02-May-2019].

[7] Services, E. I. (n.d.). Salary Averages. Retrieved from https://ece.illinois.edu/admissions/ why-ece/salary-averages.asp

[8] IEEE Code of Ethics. (n.d.). Retrieved from https://www.ieee.org/about/corporate/ governance/p7-8.html

[9] BU-304a: Safety Concerns with Li-ion. (n.d.). Retrieved from https://batteryuniversity.com/learn/article/safety_concerns_with_li_ion

[10] How to calculate battery run-time. (n.d.). Retrieved from https://www.powerstream.com/battery-capacity-calculations.html

## Appendix A Requirement and Verification Table

The R&V tables for all components are provided below. We also give the current verification status of these components to show whether or not these components have been verified properly.

**Table 4 R&V Tables**

| Component | Requirements | Verification | Verification Status |
|---|---|---|---|
| RGB Webcam | 1. Be able to capture frames at speeds of 720p/30<br>2. Image quality of camera is usable for each frame | 1. Verify the camera is receiving data at 30fps through pythons openCV library<br>2. Display the frame data and make sure that the image is not oversaturated or dark. | Yes |
| Hand Modeling Pipeline | 1. Receive 3D hand data at speeds of >10fps | 1. Verify that the data being received is correct using python particle viewer<br>2. Add a timer to pipeline to count the fps of inferencing | Yes |
| Unity Interface/ Plugin | 1. Power the microcontroller for at least 2 or more hours | 1. Make sure that we are passing the formatted data from the headset quick enough for the data. Verify that the data is accurate by comparing the data sent from the python server to what we are receiving in the Unity game engine.<br>2. Create a Unity game demo with basic hand movements. To verify the joint segmentation is working, check that all 10 fingers can move independently as well as each of their joints. Now check that the depth works by moving your hands towards various targets within the game at 3 depth levels at varying speeds. Finally, check the latency of the movement of the hands by setting a timer in the game. Have a test where the game prompts you to move your hands and times the length at which you take to respond. We are expected responsiveness to be < 0.1 seconds. | Yes |
| Python Server | 1. Send data over LAN to the Unity demo game<br>2. Receive vibrational codes from the Unity client | 1. Use the unity demo game to verify that the data is being sent.<br>2. Again, use the unity client to receive codes from the Oculus | Yes |

| | | | |
|---|---|---|---|
| IR Transmitter | 1. Eight different codes must be able to be transmitter from the Raspberry Pi to the bracelets | 1. This can be checked by sending all codes from the headset to the bracelet and verifying the codes are correct on the microcontroller. | Yes |
| Adafruit Trinket M0 | 1. Correctly reads IR codes from IR receiver<br>2. Can control the motor at three different speeds<br>3. Can control the LED to display codes | 1. Verify all four codes are received and correctly read from the headset<br>2. Check that the 2V, 3V, and 3.7V outputs can be reached for the motor from the pwm output pin<br>3. Make sure that the pwm output for the led can flash the LED whenever a code is read from the receiver | Yes |
| PWM Circuit/ LED | 1. Verify that the motor can receive 2V, 3V, and 3.7V from the microcontroller. | 1. This verification step can be done using a voltmeter and some simple code from the microcontroller to test each voltage level | Yes |
| IR Receiver | 1. Receiver is able to read all IR codes from the headset and relay them to the microcontroller | 1. This can be checked by sending all codes from the headset to the bracelet and verifying the codes are correct on the microcontroller. | Yes |
| Power Supply | 1. Power the microcontroller for at least 2 or more hours | 1. Allow the bracelet to run while receiving haptic feedback codes somewhat frequently. Time the amount of time it takes before the battery on the cell dies. | Yes |