Final Report - Team 10

Austin Fernalld, CSE, Connor McLaughlin, EE, Alex Smith, CSE, Alex Bonstrom, CSE

FIVR: Finger Interaction with Virtual Reality

***Abstract*—Virtual reality gaming is a rapidly growing market all over the world. While many advancements have been made in recent years to improve the visual and audio quality of VR environments, there haven't been many developments to improve physical interactions in virtual environments. At FIVR, we are in the process of designing a virtual reality controller that is able to provide feedback based upon a user's hand and grip position, allowing for a more lifelike gaming experience than ever before when interacting with in-game objects.**

I. INTRODUCTION

The technology upon which today's gaming systems are built is improving at a rapid pace, and due to this progress, the world has games that are more interactive and immersive than ever. Graphics have gone from 8-bit to 4K, providing an incredible visual experience, and audio has gone from mono to immersive surround sound, but the controllers used to play these games have not advanced at the same rate. Most gaming controllers today follow the same format they did 30 years ago: a plastic handheld device with a multitude of buttons and joysticks.

The gaming world has put forth one major trend in recent years: virtual reality. It is now possible to enter the world of the games that have been created, but the 'reality' aspect begins and ends with vision. The controllers used to manipulate the environment are still based primarily on button input. After an initial push into movement-sensing controllers with the wildly successful Nintendo Wii, controller innovation was largely ignored for some time. Recently, Oculus have released their Touch controllers for their Rift VR system, which claims sub-millimeter 6 DOF (degrees-of-freedom) tracking [1]. Still, there is the issue of feedback. With these controllers the user is holding onto a plastic stick with buttons, and only given feedback in the form of vibration.

Our proposed solution is defined by the ability to design a mechanical controller that has a great deal of freedom of movement while allowing the controller to provide force-based feedback to the user depending on which object they are attempting to interact with in the virtual world. We have decided that our solution must determine the hand and individual finger positions of a user's hand, map this into the virtual world, and provide feedback at a multitude of different grip positions with latency quick enough to maintain the immersive feel of virtual reality, and weight that is less than or equal to three current gaming controllers (which are rather light). Power is not a major consideration of our project, as we are using a tethered design to minimize the latency. Further detail on our proposed system specifications is provided in Table 1.

Specification	Value
Weight	<1.8lbs
Finger Position Accuracy	+/- 5mm
(X, Y, Z) Position Accuracy	+/- 7.5cm
Distinguishable Finger Locations	10 positions/finger
Latency	<100ms

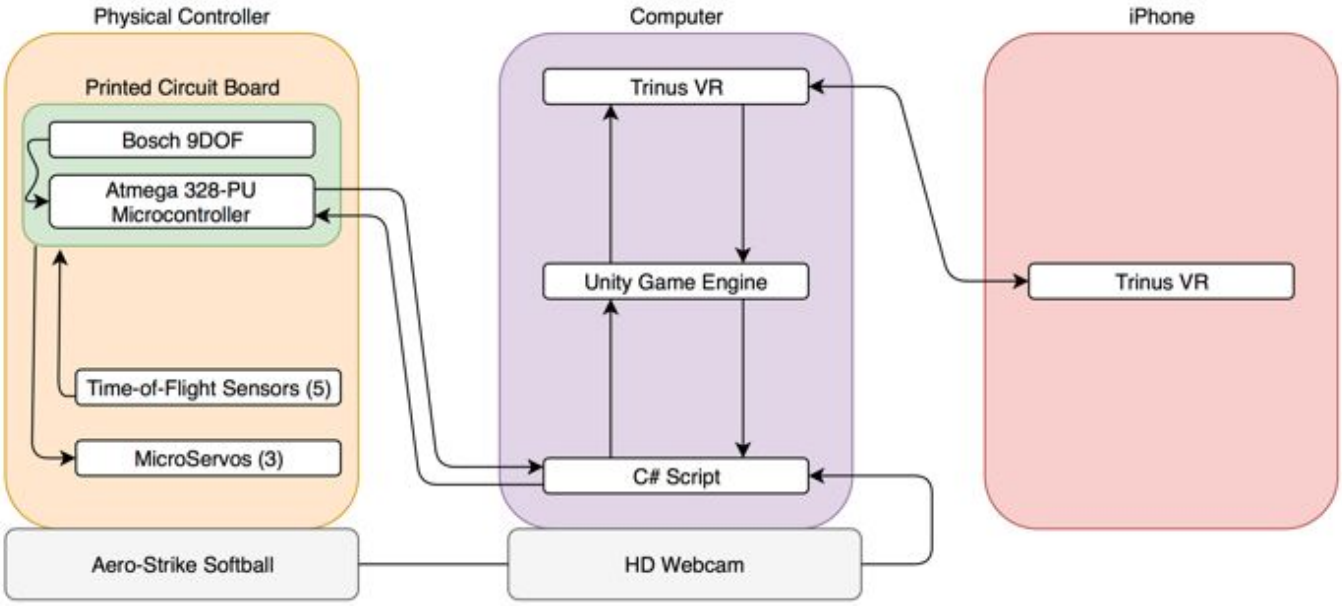
Table 1. System Specifications

II. DESIGN

A. Overview

With FIVR (Finger Interaction to Virtual Reality), we are attempting to solve the problems of controllers from the past with a new design approach. Our controller will determine the exact position of a user's hand in free space, the current position of each of the user's fingers, and provide resistance when the user attempts to grip down on an object inside of a virtual reality environment.

Some technologies which will be utilized to achieve this goal include the OpenCV image processing library, C# programs, a microcontroller for reading in and managing all data types, a modern cellular telephone for displaying the virtual reality environment, and a host computer as the main hub for interaction between our multitude of system



components. For the feedback portion of our design we will be utilizing micro servos to stop the lightly-pressurized springs which are moved by each finger, and a time of flight sensor to measure the current compression of each spring, which corresponds to the current position of each finger.

The main four blocks in the diagram include the controller, the computer, the webcam, and the phone. The controller houses one position sensor and one micro servo for each finger, the gyroscope and accelerometer (integrated in one unit) for sensing rotation of the hand, and additionally the tracking object is attached at the base of the controller near the palm of the hand. Our fast microcontroller and wired communications channels will allow for low latency between action and response. The micro servo at the base of each finger will allow for multiple locking positions to simulate grip feedback. A minimum of 10 positions on the servo to be represented in the VR system.

After research into the average weights of video game console controllers and the longevity of playtime we selected our controller to be less than 1.8 lbs. [2] Offloading the power supply to the computer allows the weight of the controller to be minimal, and the accuracy of our sensors and information collection will ensure that finger position and hand position are read in without error. With complex dimensions and shapes needed to mirror, each finger will operate on its separate servo to provide a more accurate representation of objects. When choosing a measure of accuracy and the capabilities of the TOF sensors we selected a finger accuracy to about +/-5mm. As small as a margin as that is we provided believe it is highly possible in our system.

The computer will act as the main communication hub between all components of the project. The webcam will read in live data to determine the user's hand's current location relative to the body. The phone will run the Unity game engine and display the virtual reality environment interacting with the controller motion.

A. Hand Orientation and Movement

This block will provide hand position data in the form of (x, y) coordinates and a radius value that will then be passed through the microcontroller to the Unity game engine [3]. In order to track the hand position, we will be using a bright yellow Aero-Strike softball [4], the color of which has unique HSV (Hue, Saturation and Value) values to ensure that our program will be able to discern the correct object to track by its color and shape. The center of this object is tracked via (x, y) coordinates and the boundaries of the color are used to determine the current value of the objects radius, in terms of pixels. This object is also extremely lightweight and has virtually no air resistance.

A webcam will take in data from a fixed distance away from the user, and the frames from this webcam will be passed through an OpenCV C# script which takes existing OpenCV frameworks such as motion detection and tracking and utilizes them in a C# script. While OpenCV is normally made in Python, our team used EmguCV, a C# wrapper for OpenCV. The (x,y) coordinates of the center of the softball will be updated over thirty times per second and sent to Unity using a Python script. This data will be passed through the USB interface to the computer in real-time, where this information will be used to determine Unity VR coordinates. Additional data provided from an accompanying Bosch 9DOF sensor [5], which includes a 14-bit accelerometer and 16-bit gyroscope, will be interfaced via I2C communication and sent into Unity to determine the user's current hand orientation.

The user experience will be the most crucial component of testing this block's effectiveness and accuracy. If there is a noticeable difference between where a test user believes their hand should appear in the VR game and where it is actually displayed, this will interfere with the overall success of the project.

The Bosch sensor seemed promising but several setbacks in the project resulted in the sensor being turned off during our project demonstrations. We first had some trouble calibrating the sensor when the sensor was on our prototype breadboard since the calibration routing required multidirectional movement. Furthermore, the datasheet was very unclear as to what starts the calibration and what will lock in the calibration values. They were unclear because the system does most of the work for you. We needed to put the sensor in CONFIG_MODE and then start performing the calibration routine until a status register changes to all "1"s. Once we figured that out, we started to make use of the sensor for tracing Z axis (front and back) and rotational data. Everything worked smoothly sending the data to Unity until the sensor is tilted passed 90 degrees to any one side. At this point the output in Unity would go haywire and the hand would violently twist and turn with nothing being able to stop it besides a game reset. We were not really sure what was causing this issue and weeks of debugging did not reveal a suitable cause. Talking to online communities led us to believe that the way Unity interprets the float values may be different than the way we output the the Bosch sensor data. After 90 degrees the sent data is most likely unreadable to Unity and this caused the issues.

B. Unity Game Engine

To build our virtual reality environment, we are using the Unity game engine. Unity is the industry standard video game engine [6], and it has all the tools necessary for our project. The project will be developed for smartphones, specifically the Apple iPhone. Because Unity does not come standard with iOS development packages, we will be using the Google VR SDK for Unity [7], which is under the Apache 2.0 License. The package requires at least an iPhone 5 or higher that is running iOS 8 or higher. For development and testing purposes, Alex Bonstrom has volunteered his iPhone 8 that is currently running iOS version 11.1.2. The phone will receive controller input signals from the computer through a USB connection. Fig. 2 below depicts a simulation of individual finger articulation within Unity.

Initially, we were setting the positioning from OpenCV and the readings from the fingers directly and instantly. This was causing a lot of a lot of jittery behavior, the hand would jump around and the fingers would dance uncontrollably. To fix this, we implemented the Math.Lerpf() function in Unity to interpolate the values. This gave us much a smoother implementation.

All communication to Unity from the microcontroller and OpenCV was handled by a separate C# application. The C# would create several connections within the host computer. First, the application opens a serial connection with microprocessor through one of the host computer's COM ports. Then the application creates a UDP client connection the host computer's UDP server. Two of these connections are created, one for the OpenCV communication and one for the finger readings and orientation sensor data. We initially used TCP for this communication, but the connection wasn't sending data quickly enough for our needs, the game engine would be several seconds behind the user at least. UDP better suited our needs, we were just concerned about speed rather than precision.

Figure 2. AutoDesk Maya Screenshot

Configuring the Unity environment requires a combination of scene and object modeling as well as scripting. The scene and object modeling is done through the Unity editor. Unity offers many tutorials online, and we used these to determine how to proceed with our scene and object development. Scripts in Unity allow any in-game object to be controlled through any input. All scripts in Unity must be written in C#. While none of us have direct experience with C#, it is an object-oriented language similar to Java, which we do have experience with. While the Unity engine has everything needed for realizing a full virtual reality environment, it is not well-suited for creating custom objects for the game. To model all the components for the game, we will be using the AutoDesk Maya modeling tool [8]. Maya is a great tool to use with Unity because Unity accepts Filmbox (FBX) files that Maya projects can be exported as. In order to conduct testing we will be using tools which Unity provides for free in their online marketplace. Since testing is very difficult to do for modeling, we will only be running unit tests for the C# scripts created in Unity.

C. Finger Positioning

The controller block is responsible for sensing the player's hand position in addition to providing haptic feedback by restricting the player from closing their hand. Finger sensing is

done with the use of five small, low powered, time-of-flight (TOF) sensors called the VL6180x [9]. Their job is to measure the distance between the finger and palm. The TOF sensors communicate with our Atmega328 microcontroller through the I2C interface. The Atmega328 transmits the distance values to a computer via serial to USB port which then gets sent to the phone by KinoVR. The in-game finger model moves in response to these distance changes.

Finger sensing is accomplished with the VL6180x sensor made by STMicroelectronics. However, our project uses five breakout boards made by Pololu which incorporates the VL6180x and can be seen in figure 3 [10]. This breakout board was chosen over the standalone sensor because of the ease of system integration. The VL6180x sensor by default is a surface mount component that is 4.8 x 2.8 x 1.0 mm in dimension [11]. The process of using a reflow oven on such a small component was not worth the $25 saved by forgoing the Pololu breakout board. Another reason for sensor selection was that the breakout board by Pololu featured a voltage regulator which made integrating the 2.7V device on our 5V power supply plug-and-play [10]. All that was required to add the VL6180xs to our breadboard prototype was to solder 7 pin straight headers.

Figure 3. Pololu's breakout board with accompanying VL6180x shown beside a quarter for size reference.

The VL6180x is able to consistently take range measurements from 0-100mm with a resolution of 1 mm [11]. Our team measured the distance our fingers travel from open palm to picking up a shot glass and found that our fingers did not move more than 100mm. Therefore, our finger measurements should never fall outside of the recommended accuracy range. The VL6180x's datasheet reports that measurements can be taken up to 200mm with some loss in accuracy.

The sensors were tested to see if they fulfill the first requirement in our system specifications of measuring finger position within 5mm. We tested the accuracy of the sensors by conducting five tests at varying distances with 5 measurements taken from each of the four VL6180x. The arbitrary distances are based off objects in the lab that provided a steady base to lay a low reflectivity object across. This yielded tests that had a target distance of 122mm, 88mm, 64mm, 29mm, and 17mm. The test results are show in Appendix A and show that without calibration the sensors are inside the 5mm threshold when we average out the 5 tests. However, there are some measurements that are 10mm outside the target distance which could create a bad visual effect for the user who will see their fingers jump briefly to another location.

The accuracy of the TOF sensors was not a problem in the final project. The refresh rate of the system along with Unity interpolating those changes in finger position over multiple frames caused the spikes seen in the test conducted to be canceled out by incoming readings. Erratic TOF sensor distances were not able to move the finger objects fast enough before the system fetched a more accurate measurement.

The sensor is able to take continuous range measurements at a range period equal to about 11.5ms. This was found using formula one shown below:

$$RangePeriod = Precalibration + RangeConvergence + ReadoutAvrgTime \quad (1)$$

Pre-calibration is the time needed for the sensor to calibrate it's instruments before each reading which lasts about 3.2ms. Readout averaging is used to reduce measurement noise and is set to 4.3ms by default [11]. Range convergence time is largely influenced by the range and reflectivity of the object it is sensing. The max convergence time at 100mm for an object with 17% reflectivity is about 3.69ms so we will use 4ms to be safe [11]. This gives us a worst-case readout period of 11.5ms, which translates to a theoretical finger sensing rate of 87Hz. This would be more than enough to give the user the 60 frames per second needed for smooth gameplay.

This theoretical sampling was obtained by our implementation but we did not end up using this sampling rate because it was too fast. We found that sampling at this rate caused either Unity or the C# serial port script to stall and not update the finger game object. Lag spikes were seen in the game every couple of seconds when there was large amount of finger movement. For this reason we introduced a 19 ms delay into our implementation which can be seen in Fig. 4. The orange row is the output onto the serial port which has a period of 33.47ms or 29.9Hz. The blue row is the I2C's data channel (SDA) and during the first block it tells the TOF sensors to take a measurement. A 19ms delay is then put in between this first block and the next block, which fetches the outputs of the measurements. This gave the sensors an extra 6.5ms to complete their calculations so we were never fetching the output prematurely. Another delay of 7.5ms was added in between obtaining those results and actually writing those results out to the serial port. This along with the time needed to transmit messages on the I2C bus gave our system a

refresh time of 33.47ms which is about 30Hz. While this didnt meet our specification of 60Hz, given a different game engine with better 3rd party controller support we believe this problem would have disappeared.

Figure 4. Data Output.

D. Micro Servo & Feedback

TOF Sensors are positioned to record each finger's distance to the palm to accurately draw the hand in the VR system.

Digital Metal Gear HV Servo (Corona DS-319HV): The micro servo component of FIVR is utilized to provide feedback to the handheld controller by locking the internal gears at a set positional degree given by the microcontroller. The servo operates on a 20ms period, corresponding to a frequency of 50Hz. Communication with the microcontroller is possible by first converting the CLK frequency to the desired 50Hz in the Phase Correct PWM Mode, shown in formula 2 below [12]:

$$f_{OCnxPCPWM} = f_{clkI/O}/(2 * N * TOP) \quad (2)$$

The selected Phase Correct PWM Mode operates at a high-resolution phase and frequency correct waveforms. It is based on a dual-slope operation that begins at BOTTOM(0x00) and starts up-counting until it reaches TOP (value selected in frequency equation). While up-counting and a compare match between the timer and our selected value(OCR1x), the output is cleared. While down-counting and there is a compare match between the timer again then the output is set. The output would look similar to Fig. 5 below[13]:

Figure 5: Timing Diagram

With full control over the PWM we underwent experiments to test the compare values to the servo's positions. By testing maximum and minimum pulse widths we can record the max and min angles for our micro servo. Implementation with the controller dimensions is necessary to assure the hand movements are correct in each setting.

The value chosen to alter the pulse will be reliant on the VR data and a collision flag has been set. Depending on the dimensions of the VR object the controller is coming into contact with, the value(OCR1x) will be set accordingly to which the servo will lock in a position to mimic the dimensions of the object. With a servo for every finger the dimensions for each could vary depending on the object. For example, the grips of a broom or a baseball are different for each finger specifically so by applying a servo for each would provide a more accurate representation of the objects.

The angle the servo moves will be related to the feedback given to the controller. When receiving the proper information for the dimensions of the object the servos will react accordingly and bend to the proper amount.

After continuous modifications to the mechanical design to where to place to micro servos we underwent some complications. With some fluctuation of PWM when first starting up the controller the servos occasionally receives an incorrect data which results in the arms of the servo over extending their range within the controller dimensions. When the servos receive this PWM that is out of their range results in the gears of the servos grind and pushing pressure on an immovable object. This raised complications for our design in which we had to make sure no noise or other interference would allow even a single incorrect PWM which could result in broken servo gears.

After the completion of the mechanical design of the controller and firmly attaching the servos our design team ran into another problem relating to the servo’s power. From prototyping and testing only using one or two servos at a time to properly implement the code lead to miscalculations of power and current. The servos draw an immense amount of current compared to other components of the controller which became a complication not foreseen until it was too late. The power source used for the controller couldn't supply the proper amount of current to each servo resulting in sluggish movements or even no movements at all. This power complication was also related to some issues when the PWM would randomize and jump to a destination out of the servo’s provided range. The power source was able to provide the proper current and power for a single servo but with the use of three servos in our design, as well as the other components of the controller, the source couldn’t provide the proper current to every part. If we had encountered this problem sooner, we

would have seperated the power lines for the sensors and the servos. We would then tried to implement a capacitor bank on the servo's power line and time the servo movement so that any excess power draw could be assisted by the capacitors and and not stall.

E. Computer

The host computer will act as the hub of communication between all devices, taking in all webcam and sensor data, process this information, and then send a direct visual link to the iPhone using KinoVR.

F. Phone

The phone will run the KinoVR application which will project the location and movement of the hand. Additionally, the phone displays individual motion of each finger and entire hand movement around the body.

G. Printed Circuit Board

In order to miniaturize the somewhat messy and cumbersome breadboard implementation, a Printed Circuit Board (PCB) was designed to fit within a 4.38" x 1.38" space within the controller design. The PCB was designed to include the Bosch BNO055 orientation sensor, the 12-pin voltage shifter, the Atmega 328 microcontroller, the necessary resistors and filter capacitors, as well as the appropriate header pins to connect the time of flight sensors and micro servos. Additionally, an on-off power toggle switch was added to allow power on the board to be easily reset.

This PCB was designed in Altium's Designer software using a license provided by the school. A schematic symbol for each component was created with the appropriate pins and pins number designator, and these parts were linked together in the schematic the same way that they were connected in the final functioning breadboard implementation.

Once the schematic was finalized a footprint symbol was created for each component, which determines the connection type and physical layout of the component or connectors. The footprint symbols were linked to the schematic, and the components were then netlisted onto the board. The parts were then rearranged for their optimal fit onto the board, with an attempt to minimize the crossing of wires designating net connections. After several iterations of placement as new physical constraints of the controller were discovered, the connections were routed between the nets. The board was designed as a 2-layer board for quicker turnaround time and lower cost, meaning the power was poured around the connections on the top layer, and the ground was poured around the connections on the bottom layer, as opposed to a 4 layer design which would have allowed for power and ground to have separate layers from the 2 connection layers. Since ground is the most important layer to not have broken up too much, all the connections that could be made on the top layer were created first, and any remaining connections were made on the bottom layer. Fig. 6 below shows the final PCB layout with the component outlines, net names for each pins, and the appropriate connections between the nets (red for top layer, blue for bottom layer).

Figure 6: PCB Layout

Another important consideration was to not "suffocate" any power or ground connections, and to ensure that they had sufficient access to their appropriate pours. A design guideline of 20-25 mils of clearance per component was discovered, and so some connections were moved to allow all pins their appropriate clearance. Silkscreen pin 1 markers and part designators were created as well, but these were not able to be printed onto the final PCB due to time constraints. The PCB was ordered as a tin finish with no solder mask or tin finish, but implementation of the PCB into the controller was unsuccessful prior to demo day. We suspect that there may have been a soldering issue, as there were ~60 connections to be soldered in a short amount of time, and the leeway for error (clearance between pins) was minimal.

H. Controller

The controller needed to be able to allow the TOF flight sensors to be below where the fingers would be moving. In addition to this the controller had to accommodate the microcontroller, five wired connection for each finger, a voltage regulator, and five wired connections for the Bosch sensor.

We built the controller around a wooden dowel which allowed us to glue, drill, and even stable objects to it throughout the controller development. An aluminum structure was then built out of the dowel to allow a platform where the TOF sensors assembly could be placed. Each assembly included the TOF sensor and an aluminium rectangle that is attached to the metal structure via washers and a lock nut. The lock nut was loose enough to allow movement of the sensor for on the fly tweaks of the orientation.

Figure 7: View of the final controller setup

We originally planned to have a each TOF have a spring around the edges of the sensor to keep the fingers connected to the sensor and to provide a little feedback. However, we were unable to find a spring supplier that would make 5 custom springs for a cheap price. We were not willing to buy the minimum quantity of 200 springs so we opted to using a plastic finger alternative which had which push the fingers to the open position using its tensile strength. This worked very well but now we were having the problem of four plastic fingers on the top of the controller to only (the thumb) on the bottom. This was making it so if the user were to close their hand, the thumb will automatically close because the strength of the four plastic fingers greatly outpowered the one thumb. Our solution was to reinforce the thumb spring by adding a metal bar to the controller that would sit below the user's wrist. When the top four fingers pressed down then the force would be transferred to the metal bar which would dissipate it through the wrist. This allowed the user to control the thumb separately from the four fingers on top.

Figure 8: Back view of the final controller

The PCB was suspected not to be ready in time or broken for demo day. We therefore took actions to mount breadboard onto the user's arm to help with the weight. Extra long wire harnesses were created and attached the TOF/Bosch sensors to the microcontroller. The servos were attached to the side of the aluminum structure and an arm band was hot glued to the bottom of the breadboard to give the user something to attach it to themselves. This proved to work well for demo day and it withstood the punishment of countless demos over the course of the two days. The only issue we had with the physical controller was the entire right side of the main power line in the breadboard suddenly broke. It took Alex Smith an hour to figure out the problem and reroute the connections to another working part of the breadboard.

I. Project Management

For our FPR deliverables, we proposed that all of our components would be working correctly and integrated properly. While we were able to get most of the project integrated, we had some issues with the servo implementation as well as the orientation sensor implementation. Below is a table depicting our proposed deliverables and whether or not we accomplished it. For the last entry, we were able to implement the webcam tracking but not the orientation with the gyroscope.

Proposed Deliverable Description	Completed?
5 TOF sensors operating simultaneously	✓
Unity hand model mimics controller finger movement	✓
Demonstrate that virtual objects can be picked up by a virtual hand	X*
Micro servos move to position when near Unity object	X
Demonstrate (X, Y) tracking with webcam, and orientation with gyroscope	✓/X

Table 2: Proposed FPR Deliverables. *We did not have this deliverable ready for FPR, but it was ready for demo day.

The team has worked well together, and we have attempted to play to each other's strengths and help each other out as much as possible. A considerable amount of work has been done jointly by Alex S. and Connor who are collaborating on the mechanical design and the interactions of the TOF sensors and servos. Austin and Alex B. also formed a subgroup to work on how Unity will mimic the precise position and orientation of the user's hand based off the webcam and controller sensor data. Before MDR, we concentrated mostly on the individual components and refinement of those parts. After MDR, our team worked collaboratively to integrate all the parts together.

Each team member took initiative to learn something entirely new for each process of the project. Changing pace from rigid lecture-based structure to an independent type study gave us the opportunity to progress our design and researching skills. Each week new issues arise from the design, but our team members have been able to overcome every obstacle encountered. Alex S. had never worked with time of flight sensors or on an intricate mechanical design, Alex B. had never worked with the Unity game engine, Connor had never worked with micro servos and Austin had never worked with image tracking scripts. For each member there is a new area to gain knowledge in.

Table 3: Cost for 1 vs. 1000 Controllers

IV. Conclusion

Since MDR, our team made some fundamental changes to our implementation. We switched from using the GoogleVR SDK to using KinoVR as the virtual reality tool. Our team had neither an Android phone, nor an Apple computer on which to program on, making iPhone App development very difficult. KinoVR allowed us to emulate a VR environment on an iPhone, while the GoogleVR SDK required the Apple-only application XCode. We also switched from using TCP to UDP for our inter-system communication. This allowed for faster communication and less latency.

Currently, the motion tracking brings in the (x,y) coordinates of the object, but cannot accommodate for the z position (depth), or any rotation about the axes of the controller. We have multiple functioning Time of Flight sensors which send data to the microcontroller, which is then forwarded to a serial port in the host computer. These TOF sensors operate at about 30Hz which is half the target refresh rate of the system. However, the game is very responsive and the slower refresh rate of the TOF sensors were masked by the use of the LERP functions in the game engine.

The controller allowed for basic movement of the hand but does not give the user total autonomy. Due to limited resources and experience with mechanical designs, the final product was really only able to simulate finger movements in which the fingers are relatively straight. However, we found that once the VR headset is on the experience feels as if you are actually holding the object even though your hand is not in a position that is actually gripping the object. We think this is due to the wooden dowel pressing up against the palm while the plastic fingers push back on the user's fingers. It's possible that the pushback is masking where the fingers actually are to the game, and when there is only the game in view, it feels as if your actually holding the object.

Overall, this controller would be usable if the mechanical design was a improved and if the Bosch issues could be figured out. The core backbone of the controller works well and could have potential in the marketplace given more development. Our team is glad we could take the chance to explore this budding industry and try our hands at the future of video games and exploring amazing places.

Acknowledgment

We appreciate all of the assistance and advice that our advisor, Professor Kelly, has provided us with. We are also grateful for the constructive feedback provided by Professor Krishna and Professor Goeckel following our PDR, MDR, CDR and FPR presentations. Professor Hollot and Professor Soules have also contributed greatly to the development with component recommendations.

References

[1] Oculus Rift vs. HTC Vive vs. PlayStation VR. Retrieved February 6th, 2018 from http://www.tomshardware.co.uk/vive-rift-playstation-vr-comparison,review-33556-6.html

[2] Best PC Game Controllers 2018. Retrieved February 6th, 2018 from https://www.tomsguide.com/us/best-pc-game-controllers,review-2776.html

[3] Unity 3d Game Engine. Retrieved February 5th, 2018 from https://unity3d.com/

[4] Amazon. Retrieved February 5th, 2018 from https://www.amazon.com/Franklin-Sports-Aero-Strike-Plastic-Softballs-Pack/dp/B0064I2WD0

[5] Bosch BNO055. Retrieved February 6th, 2018 from https://www.bosch-sensortec.com/bst/products/all_products/bno055

[6] "This engine is dominating the gaming industry right now", The Next Web. Retrieved February 6th, 2018 from https://thenextweb.com/gaming/2016/03/24/engine-dominating-gaming-industry-right-now/.

[7] Google VR. Retrieved February 5th, 2018 from https://developers.google.com/vr/.
[8] Autodesk Maya. Retrieved February 5th, 2018 from https://www.autodesk.com/products/maya/overview.
[9] “VL6180x”, STMicroelectronics, Accessed February 3rd, 2018.
http://www.st.com/en/imaging-and-photonics-solutions/vl6180x
[10] “VL6180X Time-of-Flight Distance Sensor Carrier with Voltage Regulator”, Pololu, Accessed November 14th, 2017. https://www.pololu.com/product/2489
[11] STMicroelectronics, “VL6180x proximity and ambient light sensing (ALS) module,” en.DM00112632 datasheet, Sept. 2013 [Revised Mar. 2016].
[12] Google. Microservo. Retrieved February 5th, 2018 from https://embedds.com/controlling-servo-motor-with-avr/
[13] “8-bit AVR Microcontrollers ATmega328/P Datasheet Summary”, Figure 15-8. Phase and Frequency Correct PWM Mode, Timing Diagram, Atmel, Accessed February 1st, 201

APPENDIX

A. The graph below shows the result of four VL6180x time of flight sensors tested for relative accuracy of a target distance which has a +- 5mm error bar attached.