ELEC 490 Final Report

Group #33

Adel Abdel-Hamid 13aah16@queensu.ca (10142673)

Eric Chau 13ec50@queensu.ca (10141119)

Monica Rao 13mrr6@queensu.ca (10133414)

Faculty supervisor: Dr Saeed Gazor

Department of Electrical and Computer Engineering

Queen's University at Kingston

April 5, 2018

Contents

1 Executive Summary

The objective of this project is to build a Kinect-type device, named the VRSS, and connect it to a VR system in order to simulate a surgical procedure. The VRSS must be as small and precise as possible in order for it to be easily transported and for the simulation to be as realistic as possible respectively. The scope of this project will consists of a fully functional Distance sensor, a fully functional pair of sensor gloves, a fully functional virtual world, a bill of materials and a list of the codes.

This project was chosen to improve training possibilities for surgeons before working on a patient and without the need for surgical materials that are limited and can be expensive to acquire. By using the VRSS, surgeons in training can experience what it seems like to perform a real surgery without the risk of injuring their patient, and without wasting precious resources. The overall goal of this project is to build a working prototype of surgical gloves and depth sensor, paired with a working VR simulation. The hardware will mainly consist of designing and building various circuit schematics and getting them in order to assemble all components together. The virtual reality environment is planned to be programmed in Unity using C#.

A gantt chart and summary table have been made to keep the team on track for deadlines, as well as keep the team organized to schedule meetings with the project adviser. Testing and evaluating the overall project was done multiple times throughout the course of building the VRSS. The hardware was tested by checking the voltage and current values of the circuit and simulating the desired results to avoid any potential complications. The software component of the project was tested in real-time while simulating the virtual reality environment in Unity.

The overall budget for this project was planned for approximately 400 dollars, but the team already expected to exceed that amount due to the other components that were needed to be purchased for the hardware component of the project.

2 Introduction

This report summarizes the ELEC490 project undertaken by Group 33 - the VRSS group. This report has been prepared for Departmental approval. The intended audience for this report are Dr. Saeed Gazor, Dr Carlos E. Saavedra and any other interested members of the faculty.

With the growing improvements in technology also comes improvements in medicine. For the past 30 years, technology and medicine have been getting closer and closer to developing the best technology.(1) Although doctors are now capable of operating minimally invasive surgeries (MIS) and use the help of robots for more complicated procedures, fewer new technologies have been used to maximise the training and learning capabilities of future medical staff members.

The use of a VR (virtual reality) set to simulate a surgical situation allows for new surgeons to learn vital skills before taking a patient under surgery by using a computer generated environment to improve their efficiency at performing certain procedures such as MIS.(2) The use of VR training was shown to improve performance by resident doctors. In one study, residents trained with the VR simulation for a gallbladder surgery were 29% faster than their classically trained counterparts and were five times less likely to cause injury the gallbladder.(3)

In order to follow the users movements in the simulation, a motion sensor system would be used to detect the precise movements. Motion sensors such as the Microsoft Kinect have been used by surgeons before, to manipulate key medical images during surgery.(4) However, this project will connect a motion sensing glove built from scratch and a Distance sensor, known as the "D.A.V.E.S" (Distance Actuator Virtual Environment Sensor), that will be paired with a phone attached to a VR set to simulate surgery for training purposes.

3 Problem Statement

The main objective of this project is to create a virtual reality environment for doctors and surgeons alike to train and practise without the need for surgical materials. This would ultimately save resources and finances yet provide a safe and interactive way to gain experience.

With the high expense of medical equipment and with the best equipment reserved only for real surgeries, most medical students are very limited in terms of their available practise resources. Many medical students require a high number of

simulation hours of practise prior to real experience, and for a good reason. Surgery is very intensive and normally life altering for patients. However, due to the limited amount of equipment universities have available, students are required to take turns and may only use the equipment while a certified professional is present since universities cannot put themselves at a potential financial risk by having students break the equipment. With virtual reality, universities do not need certified professionals to be present as they are not risking any equipment whatsoever. The professionals would then only be required to teach, then the students may practise in their own time. This would make it substantially easier for practising medical students to acquire their required number of hours while also avoiding a significant amount of potential financial risk for the school.

Ultimately, the VRSS will save cost and time for many in the field of science and medicine along with opening up the possibility for many to experience surgeries in an immerse yet safe and budget friendly environment. Although there is no deadline which would make this implementation particularly successful, the sooner it is out in the market the better for students and practising doctors alike.

From a Technical point of view, the various components that make up the VRSS are a challenge for any engineer. In terms of hardware, the motion sensing gloves and the distance sensor must include not only extremely precise sensors to accurately simulate motion, but they must also be as light and as comfortable as possible in order to minimize movement obstructions. In terms of software, the virtual reality simulation must utilize the raw values from the hardware and accurately simulate it as closely to the real life motions as possible. In order to do so, the values must be sent over to the simulation device from the accelerometers and gyroscopes while maintaining a high level of precision (i.e. retaining decimal point values), then it must be repeatedly calibrated through testing. Only after all these steps are taken, would a feasible prototype be created.

4 Overall Design Process

4.1 Hardware Design Process

In order to fully implement the VRSS system, the hardware had to be built prior to designing the virtual reality environment. The design of the actual VRSS depended on the circuitry that the team designed and built. The team researched ahead of time a few different types of circuits to build the system, one will be for the servo motor to control the shifting of the distance sensor. Unfortunately, that will be explained later in the report that the eagle PCBs schematics couldn't be implemented. The other circuit was for the various sensors required. The team designed schematics of the two

important circuits needed, for these Eagle PCB designs, these can be seen in appendix (please refer to Appendix A). The rest of the hardware was assembled together to a full model of the VRSS system once all the circuitry is made and tested.

The system as an entirety has very simple inputs and outputs. The input being the user themselves, by using motions to interact with the surgical gloves. The output would simply be the entire virtual reality environment. Specifically, the user's movements are taken as inputs for the VRSS C# code and then the distance sensor's motor to the required position. Similarly, the user's movements are taken as inputs for the VR Unity code which in turn will affect both the new position of the character in the VR world and affect the users future inputs.The Flow chart in figure below shows the summary of how the user, distance sensor, and VR will interact as a system.

By having a structured approach, the team can ensure all the components required are completed or can be modified in time to ensure no time is lost. This also aids in terms of organization by ensuring all dependent components are scheduled to be completed after the independent components which they depend on, this is explained more in detail later in this report.

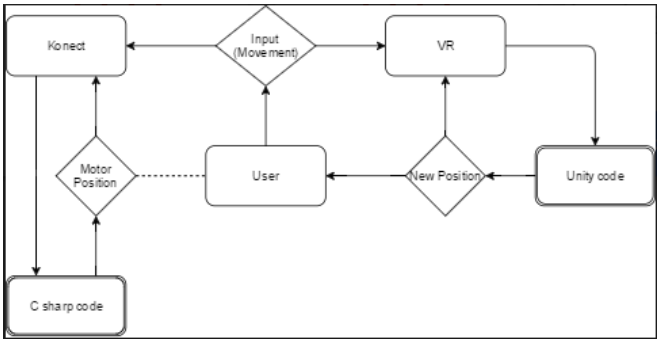


Figure 1: Flow Chart for Proposed Project

4.1.1 Original Distance Sensor Design

The original design for the distance sensor consisted of building a sensor very similar to the Microsoft Kinect. This distance sensor would have included an IR Emitter, an IR depth Sensor and a color sensor. The IR Emitter would have been paired with a laser beam splitter that would split the IR laser into a matrix as seen below. The IR sensor would then measure the depth based on infrared coded structured light (5). The IR depth sensor, which is a CCD camera captures the default matrix pattern (see figure 2) and considers it the reference plane. When an object appears between the reference plane and the camera such as in figure 3, then the matrix points will be offset and the displacement can be calculated at using the following equation:

$$D = \frac{D_0}{1 + (f \cdot b) / (D_0 \cdot d)}.$$

Where D is the distance of the point from the reference plane, D0 is the distance between the focal point and the reference plane, f is the focal distance of the camera, b is the distance between the focal origin point and the IR emitter and d is the pixel offset in the image space.

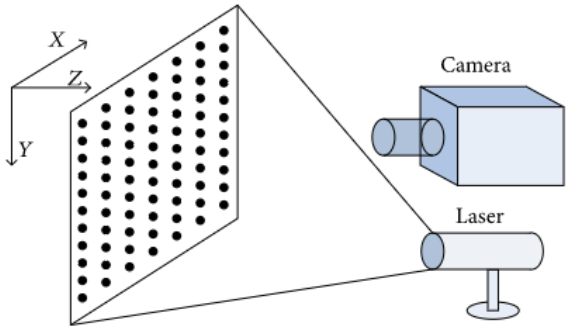


Figure 2: IR Matrix Pattern

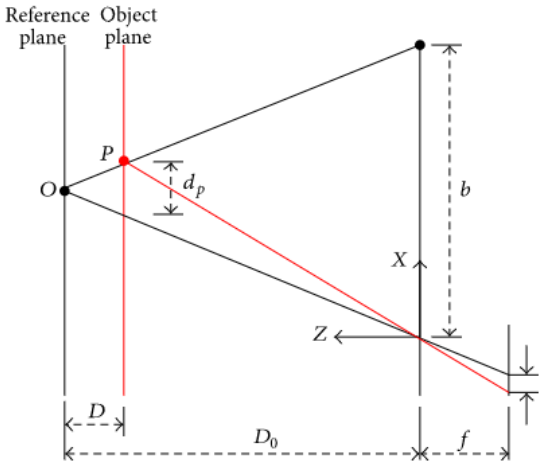


Figure 3: Pixel offset relation

The use of such a system would have made our distance sensor even more precise than our final product, however, this couldn't be implemented due to the fact that the laser beam splitter was out of stock and would take months to arrive, and because the IR laser was too expensive. As such, a simpler version of this sensor was built using IR distance sensors placed in a matrix.

4.2 Building Process

During the actually building process of the prototype of hardware the team spent a lot of time working, soldering and construction in the prototype shop in the ilc. For the distance sensor (depth sensor) construction was designed at first to have 2 moving servo motors, but due to some complications and time constraint they were not implemented in the final design.

The final constructed design of the distance sensor ended up being built with recycled wood from the prototype shop, and distance sensors that were later glued to the main base to create a stable, compact, simple unit, that has room for the bread board and Ardunio board to be connected. Figure below shows the prototype shop depth sensor. The team later named this sensor D.A.V.E.S.

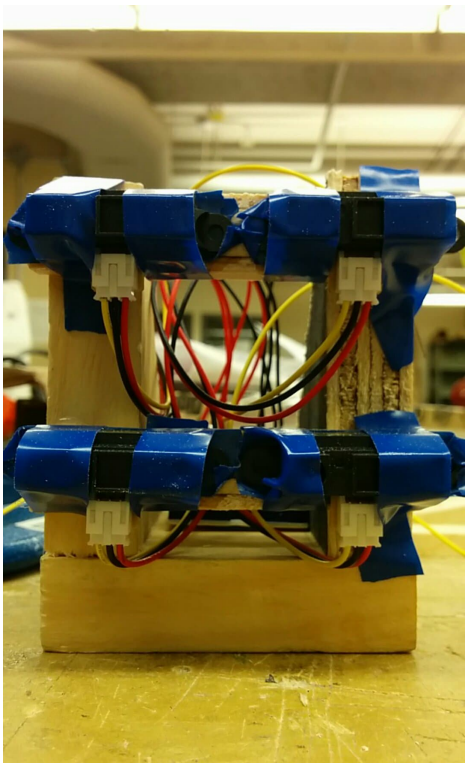


Figure 4: Depth Sensor Prototype in Workshop

The design of the surgical gloves has changed a few times throughout the course of the project. Originally, the team thought of the PCB design for the gloves on eagle, as mentioned before due to time constraint and money the team just decided to hard-wire the circuits. The sensors were glued to the gloves using epoxy glue. The figure below shows the team first putting together the right hand surgical glove and hooking it up to the Ardunio. Please refer to the figure below.

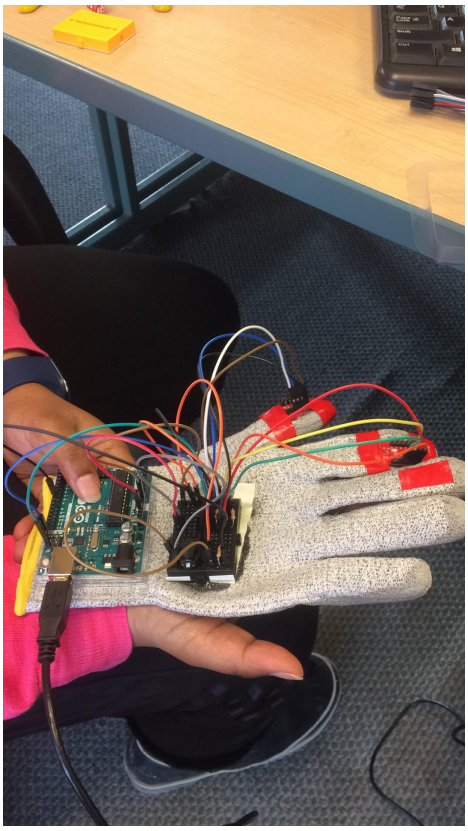


Figure 5: Protytype of Right hand surgical glove

4.3 Software Design Process

The software design process revolved primarily around the flow of data. This means from the moment values are generated from the hardware, how it is processed, and finally how it is used. Problems encountered and solutions will be explained in the Implementation and Testing portion of this documentation. The following simply shows the flow of data.

1) Hardware (i.e. gyroscope and accelerometers each generate their own XYZ values)

2) Right glove is connected to laptop USB port COM3 with a data rate set to 9600, while left gloves is connected to laptop USB port COM4 with a data rate also set to 9600 (NOTE: right and left hand Arduinos run on different scripts, due to right hand specifying COM3 communication and left hand specifying COM4 communication)

3) All gyroscopes and accelerometers are checked in Arduino to ensure proper functionality

4) Both left and right hand gyroscopes along with both index finger and thumb accelerometer XYZ values are read into

Arduino and each hand's Arduino's script stores the values into an array (total of two arrays, one per script/hand, with each array containing 9 elements)

5) All the array elements are then double integrated, due to their acceleration values, in order to produce position values

6) Each script then sends each element's value over their appropriate COM port via serial prints, with each concatenated serial print delimiting each element with a comma

7) Unity receives these values where each element from Arduino is now stored into individual float variables on Unity into two separate scripts, each representing the gyroscope for each hand

8) The gyroscope then sets the XYZ coordinate values for each hand on their designated scripts

9) Each finger for each hand also uses their own designated scripts, where each finger inherits the values from the gyroscope object

10) Finally, once the values are inherited, each script my set their own XYZ coordinate values for their designated finger

11) For displaying the virtual reality environment on our Android device, a host/slave app which utilizes the LAN's network was used

12) The host app is started on the computer which also hosts the virtual reality environment, where the host app essentially starts up the host server and requires an input of the Android device's IP address in order to search for it

13) The Android device simply starts up the slave app, where it searches for a receiving connection

14) Once the connection is established, the computer's host app sends all the information to the Android device, where the Android device's display shows the virtual reality environment and the Android device's gyroscope is used to sense rotation while the computer's hardware is used to simulate the virtual reality environment

5 Project Planning and Budgeting

5.1 Project Planning

The team set weekly meetings every week earlier on the previous year when they were in the makes of planning out their project. This made the group be able to keep track on all the group members were on track with their responsibilities to the project. The team decided when it was closer to the deadline the team set milestones for one another in order to be able to make sure everything will be done in time for the showcase. Two of the group members that mainly worked on the hardware components took regular meeting notes. Please refer to the appendix for the specific notes taken. The summary of the planning of milestones for group 33 is summarized and you can refer to table 1 below. Figure below shows the gantt chart that the team made to show the progress plan for the project over the year.

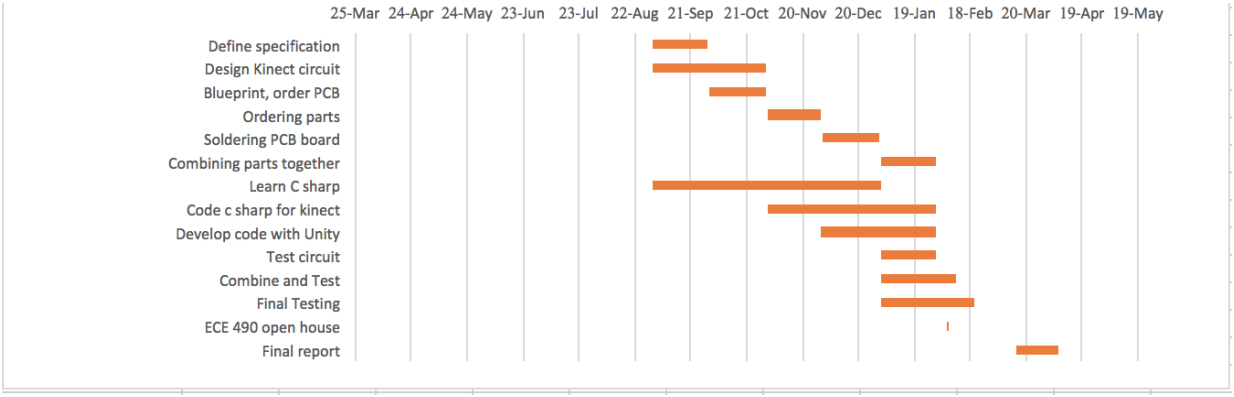


Figure 6: Gantt Chart for the Work Plan

Table 1: Project Schedule Table

Milestones	**Detail Actions**	**Starting and End Date**
490 project proposal	write a proposal report that outlines the plan for 490	April 2017
Define Specification	Define the size, cost and performance with adviser	Sept 1, 2017- Sept 30, 2017
Design Kinect circuit	Design schematic, layout, and prototyping	Sept 1, 2017 - Oct 31, 2017
Blueprint, order parts	submitted blueprint and then order remaining parts	Oct 2017
Ordering Parts for Kinect,and VR head set	Ordering off parts on-line	Nov 1, 2017 - Nov 30, 2017
Soldering PCB board	Soldering	Dec 1, 2017 - Dec 31, 2017
Combining all parts together	Design the combined schematic	Jan 1, 2018 - Jan 31, 2018
Learning C sharp	Learn the new language	Sept 1, 2017 - Nov 30, 2017
Code c sharp for Kinect	Develop code in c-sharp for our kinect	Dec 1, 2017 - Jan 31, 2018
Develop code with unity software	Develop code to work with VR	Nov 30, 2017 - Jan 31, 2018
Testing of the Circuit	Testing voltage, current,and signal outputs	Jan 1, 2018 - Jan 31, 2018
Combining and Testing	Combining testing of working kinetic,code	Jan 1, 2018 - Feb 10, 2018
Final Testing	Testing Kinetic and VR working together	Jan 1, 2018 - Feb 20, 2018
ECE 490 open house	ECE demo their complete 490 projects	Feb 6 2018
Final Report	Must write the final report of completed project	April 6 2018

5.2 Required Components

The parts that need to be purchased will need to be accelerator, laser sensor, motion sensors, few high power cameras, resistors, capacitors, inductors, one 7 pin op amp, 3 16-pin op-amps, approximately 3 external batteries, three small Ardunino boards, LCD monitor, a motor (for the leaning forward and back movements) etc. The team planned to put in a order for a custom PC boards but due to budget limitations and time constraint it was not possible to complete.Refer to Appendix for the full design for the PCB boards. A virtual relativity head set was donated by one of the team members.

5.3 Lab Space Used

The team used the prototype shop to use the soldering iron in order to solder the circuit components to the sensors for the surgical gloves. The lab space was used to test out the circuit components and measuring with an oscilloscope the wave output, voltage, and current values. This is to ensure that correct circuit layout works before placing an order to avoid any

potential disasters.

5.4 Equipment Required

The equipment needed for this project will be include the following: soldering iron from the prototype workshop, a multi-meter, screw driver, various Ardunio male and female wires and probes from the common electrical kit, a LCD screen, motor, and three small Arduino boards etc.

5.5 Software required

The software needed for this project included the following: Eagle in order to design the layout, schematic and custom PCB board designs, Unity and C#/Ardunio will be used to code the software components of the project for the VR headset as well as the custom kinetic. Matlab, may be used to test simulations of the system. Or to help with re-calibration of the system.

5.6 Budget

The budget for this project should have been around approximately 400 dollars but the team expects to go over a bit over since the parts that need to be purchased and that could add some added costs. This is also factoring in any other potential failures in the custom PCBs or with different sensors when soldering and replacement parts. In order to avoid potential failure the team has planned that the testing of the circuits and PCBs will be looked over by a supervisor or Ta in order to make sure that the board and circuits function properly. Also, extra parts planned to be ordered to make sure if a part goes missing or when soldering the board needs to be replaced. From this caused this budget plan/ estimation the team made to avoid any complications and to successfully complete the project at hand with minor implications.

The budget breakdown of the project was developed and modified many times throughout the time line of the project. Due to some replacement of certain sensors and the various changes of the the overall design the budget ended up being as close as possible, but less than the intended original budget. The main part that saved us money was the fact as the team decided as mentioned before that the cost of PCBs was not worth the time. In the end the team decided to hard wire the surgical gloves as it was the cheaper option. The A complete breakdown of the budget and materials bought can be summarized in

the table below. A more detailed blueprint of the entire budget material with taxes included. Please refer to the appendix.

1	Group# 33								
2	Capstone Project	Adel Abdel-Hamid	Monica Rao	Eric Chau					
3									
4		Item	quantity	price (CA$)	total	link	Supplier	Supplier part #	Manufacturer #	Notes
5		arduino uno R3	2	24.95	49.9	https://www.sparkfun.com/products/11021	sparkfun	DEV-11021	A000066	Red is for USD
6		arduino mega	2	45.95	91.9	https://www.sparkfun.com/products/11061	sparkfun	DEV-11061	A000067	Red is for USD
7		Battery pack	2	4.95	9.9	https://www.sparkfun.com/products/11222	sparkfun	COM-11222	N/A	Red is for USD
8		gyroscope	2	9.95	19.9	https://www.sparkfun.com/products/9793	sparkfun	SEN-09793	ITG-3200	Red is for USD
9		bluetooth	2	4.45	8.9	https://www.banggood.com/HC-06-Wireless-Bluetooth-Transceiver-RF-Main-Module-Serial-For-Arduino-p-80364.html?rmmds=search	Bang good	80364	N/A	
10		accelerometer	6	4.95	29.7	https://www.sparkfun.com/products/13963	sparkfun	SEN-13963	LIS3DH	Red is for USD
11		ccd camera	1	14.99	14.99	https://www.amazon.ca/XCSOURCE-700TVL-Camera-Photography-AH078/dp/B0157N4QPO/ref=sr_1_5?ie=UTF8&qid=1505847627&sr=8-5&keywords=ccd+camera	Amazon	N/A	N/A	
12		servo motor	1	18.78	18.78	https://www.digikey.ca/product-detail/en/parallax-inc/900-00008/900-00008-ND/1774454	Digikey	900-00008-ND	900-00008	Red is for USD
13		servo motor	1	7.99	7.99	https://www.digikey.ca/product-detail/en/adafruit-industries-llc/169/1528-1076-ND/5154651	Digikey	1528-1076-ND	169	Red is for USD
14		resistor	1	12.84	12.84	https://store.qkits.com/set-of-610-resistors-e-12-series-k-res-e12-velleman.html	Qkits	RES-E12	N/A	
15		16-pin op amp	4	2.2	8.8	https://www.digikey.ca/products/en?keywords=16-pin%20op%20amp	Digikey	AN17827A-ND	AN17827A	
16		7-pin op-amp	4	1.62	6.48	https://www.digikey.ca/product-detail/en/texas-instruments/LM380N-8-NOPB/296-46516-5-ND/6274	Digikey	296-46516-5-ND	LM380N-8/NOPB	
17		MM FM FF wires	1	6.97	6.97	https://store.qkits.com/jumper-cable-set-m-m-m-f-f-f-20cm.html	Qkits	40PIN20PK	N/A	
18		Gyroscope	2	9.95	19.9	https://www.sparkfun.com/products/13339	Sparkfun	SEN-13339 ROHS	LSM6DS3	Red for USD
19		Velcro (5 foot)	1	11.18	11.18	https://www.amazon.ca/VELCRO-Brand-Sticky-Back-3-4-Inch-5-Feet-Black/dp/B00006IC2L	Amazon	90086	B00006IC2L	Red for USD
20					0					
21		Adel' Personnal contribution	1	60	60					
22		Monica's Personnal Contribution	1	33	33					
23		Eric's Personnal Contribution	1	33	33					
24		Total			444.13					
25										
26										
27										

Table 2: budget breakdown summary excel file

6 Project Significance and Societal Impacts

6.1 Social Impacts

"SSEERP factors inform the process and implementation. Impact on range of stakeholders are considered and risks mitigated". SSEERP: social, safety, environmental, economic, regulatory compliance, professional.

When initially deciding on a project idea in third year, many factors were considered. One large factors the team considered were the social implications, benefits, and overall impact. Specifically, the SSEERP factors associated with the process and implementation of the project. Overall, we believe a real life application of this project would benefit consumers on many levels in the medical industry. It would provide an extremely safe and risk-free method of doctors in training to practise any exercise needed, along with being more resource friendly since the VR equipment can be used over and over again without the waste of single-use material. Economically, it provides opportunities for virtual reality designers to design more since ever new simulation exercise requires a new environment to be designed. From a professional perspective and in terms of meeting regulatory compliances, there would actually be less requirements since, for example, the repetitive use of cadavers in medical schools would not require a sign off. Comparatively, once the virtual reality equipment and software is officially purchased by the school, the process of repeatedly purchasing materials such as cadavers would not be required anymore and students would be free to use the virtual reality equipment on their own contingency (barring the approval of their professor or supervisor, of course). Overall, we believe all stakeholders associated with the use of a VRSS would benefit from one perspective or another regarding the SSEERP factors.

6.2 Legal Impacts

Since the software that was used to code the project was closed and only on unity there will be no licensing for our software. The hands of the unity environment was bought from a outside party for approximately 25 dollars. So with this comes with no license, this means that our work is under exclusive copyright by default. The sensors that the team used from sparkfun, had use the specific libraries that were provided by the sparkfun company to use on their products. Exclusive to group 33, Queen's electrical and computer engineering department,and professor Gazor, as the supervisor, the no other outside party has the distribute, copy, or modify the code outright.

However, with the possibility of implementing this software in other places as the VR proof of concept can be applicable to many other applications, there are possibilities that legal assistance could be obtained to formally establish copyright ownership of the code and the hardware development to further continue the project. Aside from these stipulations, there does not seem to be any other legal issues that our team would need to consider.

6.3 Economic Impacts

With society worrying about the social impacts of different types of prototypes, reducing waste into the environment is important factor to take into consideration. With using Unity and Ardunio as the free software programs, there is no environmental impact with deal with. The hardware on the other hand, as a prototype would have waste with the use of sensors. As when dealing with sensors of any kinda, they are expensive to replace when something does wrong. As the team hardwired the gloves the waste of wires and breadboards was taken in consideration. The main issues with dealing with prototypes it is dealing with a balance between maintaining a lost cost but having waste, or keep the prototype expensive but have minimum waste. As for the distance sensor, the sensors would be considered waste as well, but the body was constructed from recycled materials from the prototype shop.

In the future the team would decide to prototype the surgical gloves to be more slim fitting with less bulk in general with wires and sensors on top of the glove, and rather have them built within the glove instead.

6.4 Project Significance

As mentioned before, the team took all impacts into consideration when dealing with the overall design of the project. With saying this the project significance is to help train surgeons in order to give them a overall exposure that they would not be able to get from just learning in the classroom itself. This project has many social impacts when dealing with the VR environment and surgical gloves. It is important that we all take this into consideration.

7 Project Reflection

7.1 Meeting Specifications

7.1.1 Hardware Reflections

The main hardware issue that could have been avoided was the use of accelerometers in order to measure the fingers position rather than using gyroscopes. At the time of ordering our equipment, all of the gyroscopes that we could find were either too large to be placed comfortably on the fingers, or were a separated piece separate from a workable PCB. As such our options were to either have uncomfortably big sensors on the tips of our fingers, build our own PCB which would take a lot of precious time out of our project, or use accelerometers and integrate our outputs in order to receive our position values.

Another avoidable issue we had involved one of our servo motors. The motor used for the vertical rotation of our distance sensor couldn't handle the weight of the distance sensor and as such failed in its only job. The reason why a subpar actuator was used was due to budget constraints. The larger servo motor used for horizontal rotations worked fine, however, we are unsure if using the same servo motor for the vertical movements would help as it is quite bulky and may in fact cause the horizontal servo motor to stop working.

7.1.2 Software Reflections

One of the major issues encountered from the software aspect was the communication conflict over the COM ports for serial data transfer. The most prominent issues include the computer prioritizing one COM port over another, leading to the secondary COM port lagging in terms of connection. This led to only one glove being connected at a time to ensure there was no interference between two different COM ports. The other COM port related issue includes the need to create classes that inherit variables from the parent class that has an open serial port. If not, all the classes attempting to open the serial port would interfere with one another, causing malfunction. A solution to this issue would be the implementation of

Bluetooth. By using Bluetooth, the primary optimization would be removing the need to stay connected via USB cable. This would allow a wire-free movement of the gloves away from the computer. Secondly, there would be no COM port interference issues since all the data would be transferred over Bluetooth. With all the data being sent on the same connection method, there would be no interference of any kind.

The second major issue encountered from the use of the VRSS is the connection between the Android device and the computer over a LAN network. The current implementation uses a slave/master connection via LAN by identifying the Android device's IP address. This implementation works in most scenarios tested, however in areas where the LAN network contains high traffic (many connections), the Android device starts to suffer from significant lag. The solution for this would mesh in with the first issue explained above; utilize a Bluetooth connection. By using a Bluetooth connection, not only would it solve the issue of the spotty LAN connection, but it would combine the dependencies of having multiple connections (COM port connection via USB and IP connection via LAN) into a single dedicated connection between the computer hosting the virtual reality simulation and the client Android device. Modern Bluetooth technology is fast enough and powerful enough to support multiple channels of data while providing a high rate of data transfer.

	Specification	Target value	tolerance	Achieved Value
1	*Distance sensor*	*N/A*	*+/- 1 cm*	*+/- 1 cm*
2	*accelerometer*	*N/A*	*+/- 2 mm*	*+/- 5cm*
3	*Gyroscope*	*N/A*	*+/- 1cm*	*+/- 1cm*

Table 3: Blueprint Hardware Specifications

1	**Functional requirements**	**Specification Met (Yes/No)?**
1.1	VRSS's ability to motion and depth sense	Motion: Yes Depth: 50%
1.2	Gloves abilities to motion sense to a greater degree than VRSS	Yes
1.3	VR headset's ability to see within VR environment while simultaneously simulating movements read from the VRSS and glove sensors	Yes
2	**Interface requirements**	
2.1	Smartphone contains simulator's APK installed and ready to run	Yes
2.2	VR headset is able to hold smartphone to provide VR experience	Yes
2.3	Gloves are able to fit user, and visible within VR environment	Yes
2.4	The VRSS is able to read user's movements, shown by its ability to simulate it within VR environment	Yes
3	**Computer Performance requirements**	
3.1	OS: Windows 7 SP1 or newer	Yes
3.2	Processor: Intel Core i5-4590 equivalent or greater	Yes
3.3	Memory: 4GB RAM or more	Yes
3.4	Graphics: Nvidia GeForce GTX1050TI or greater	Yes
4	**Smartphone Performance requirements**	
4.1	OS: Android 5.0 Lollipop or newer	Yes
4.2	Processor: Snapdragon 800 equivalent or greater	Yes
4.3	Memory: 2GB RAM or more	Yes

Table 4: Blueprint Software Specifications

7.2 Team Reflection

The overall team was very satisfied with the effort put in by all team members. The team enjoyed working with each other and had a very memorable time being able to create a working prototype of the surgical gloves and depth sensor to coincide with the VR environment. With saying all of this, this justifies for the grade distribution for all the team, as everyone put in the same amount of hard work into this project. Therefore the figure below shows the grade distribution of the team. Please refer to the table below.

Name	Overall effort expended (in %)
Adel Abdel-Hamid	33.3
Monica Rao	33.3
Eric Chau	33.3

8 Implementation

8.1 Hardware Implementation

The Motion sensing gloves are comprised of three accelerometers, one on each of the thumb, index and middle fingers, a gyroscope on the backhand, an Arduino on the wrist and a reset button. All these components are connected to one another with male-male and male-female wires on miniature breadboards.

The Arduino is connected to the PC using a USB cable which provides power to the entire glove. The gyroscope and the accelerometers are all connected to the Arduino using I2C connectivity, as mentioned later in the document. Due to the I2C connection, bus capacitance has to be taken into consideration and resistors were added into the circuit in order to counter it.

The accelerometers used were Sparkfun LIS3DH triple axis accelerometer, these were small and inexpensive accelerometers that could be used to measure the fingers position without affecting its freedom of movement. These accelerometer use a 3.3V input power voltage rather than the typical 5V. Furthermore, since all three accelerometers are identical, they each had the same address (0x19) and couldn't be used in the same I2C line. In order to circumvent this issue, one of the accelerometers was switched to the alternate address (0x18) while a switching mechanism was introduced between the other two accelerometers with the same address. The switching mechanism is in fact an arduino code which does the following: instead of having the two accelerometers connected to the 3.3V input pin, they would be connected to the digital pins 3 and 4 respectively. First pin 3 is HIGH and outputs 3.3V into the accelerometer on the index finger (arduino pins only output 5V when HIGH so in order to get a 3.3V, a voltage divider circuit was built with 1kilo-ohm resistors), while pin 4 is LOW for 20ms. Then for 10ms, both pins are LOW. Finally, for another 20ms, pin 4 is HIGH and outputs 3.3V into the accelerometer on the middle finger. This would give each finger 20 readings every second which is quite acceptable.

The gyroscope used was a Sparkfun LSM6DS3 6 degrees of freedom gyroscope. This gyroscope is a bit larger than most but doesn't interfere whith the hands movement due to its position. Just like the accelerometers, it uses a 3.3V input power voltage and is connected within the same I2C system as the accelerometer. However, it doesn't interfere with the accelerometers as it has its own address (0x6B).

Due to the large amount of sensors in parallel on the bus line, the bus capacitance of the system has increased dramatically.

A high capacitance limits the maximum data rate which can be transferred over the SDA and SCL lines. In order to counter this issue, two 1kOhm resistor were added connecting the Power line to the SCL and SDA lines respectively for each sensor added after the first (ie. if there are three sensors on the I2C lines, then there should be a total of four 1kOhm resistors on the glove.

Finally, the last piece of hardware on the gloves are the reset buttons located on the backhand of the gloves. The reset button operates outside the I2C line and is a simple system. When left alone, the circuit is open and digital pin 7 receives a LOW input. However, when the circuit is closed, a current passes through the resistor and pin 7 becomes HIGH. As soon as that happens our Arduino code enters a 5 second delay allowing the user to re-position himself into the default position.

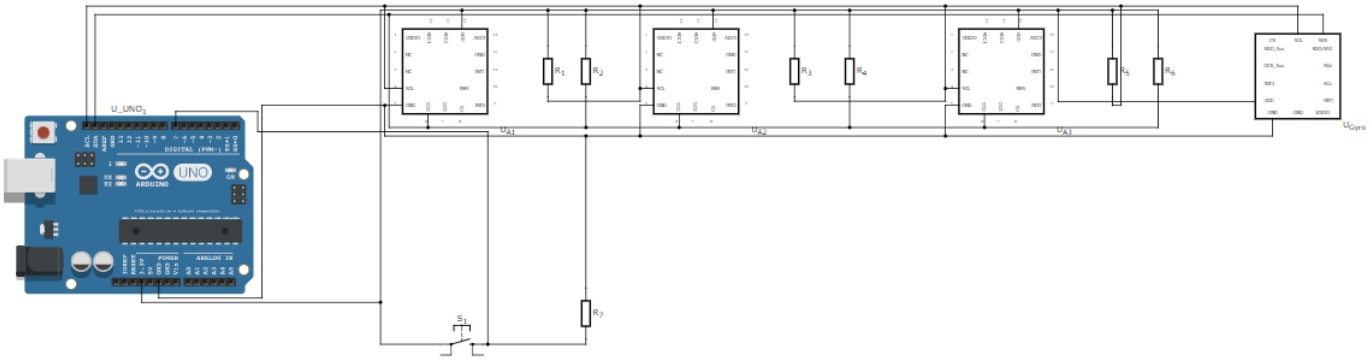


Figure 7: Glove Schematic

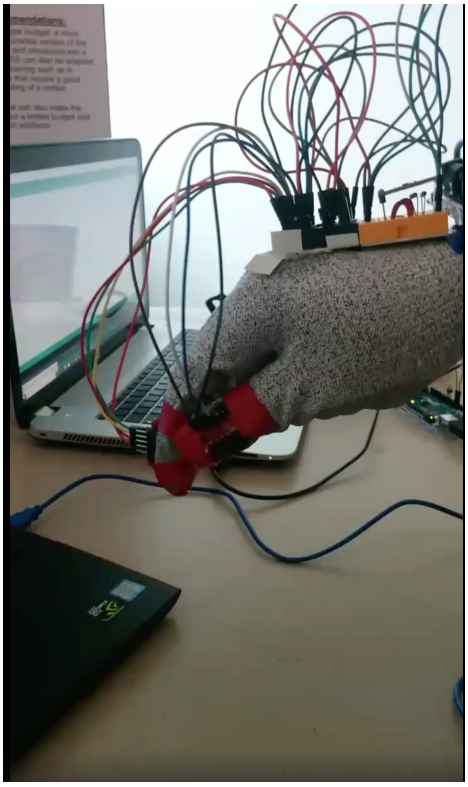


Figure 8: Gloves in use during Open House demo

The distance sensor is comprised of four IR sensors placed in a 2x2 Matrix and two servo motors, all connected to a Arduino Mega. Each IR sensor is connected to an analog pin on the Arduino and is also directly connected to a 5V source and the ground, no resistor is needed. The IR sensor senses the location of and object directly in front of it. As seen in table 5 and figure 9, the IR sensors output follows the shape of an inverted parabola with the highest output being at 6cm from the IR sensor. This means that the VR user must make sure that they do not get too close from the distance sensor or they risk confusing the system because any point before the 6cm has a value equivalent to a point after the 6cm threshold.

	Values			
Distance (cm)	BR	TR	BL	TL
3	545	250	537	430
4	640	570	640	580
5	680	650	680	650
6	660	680	670	680
7	640	650	645	640
8	580	620	600	620
9	540	550	550	550
10	500	500	500	500
12	420	440	420	440
14	360	370	360	370
16	300	340	300	340
18	280	300	280	300
20	240	280	240	280
22	220	260	220	260

Table 5: Distance Sensor Testing values

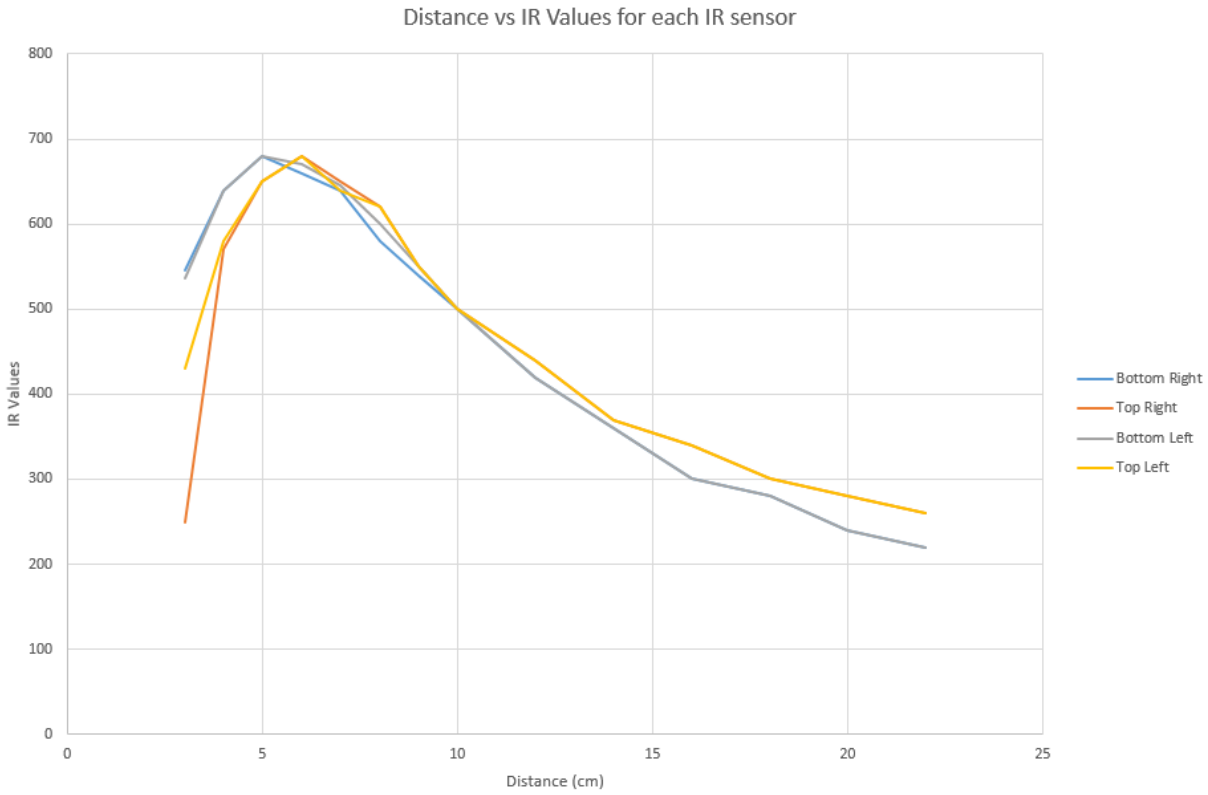


Figure 9: Distance Sensor Testing Graph

The servo motors allow the distance sensor to rotate with the movement of the user in order to realistically capture the movements of the surgeon. One of the servo motors rotates the distance sensor vertically and the other rotates it horizontally. In order to make the distance sensor move correctly, one of the hands must be placed in front of it so that all four IR sensors sense it's presence. As the hand moves around, depending on which IR sensor still senses the hands presence, then the Distance sensor will rotate accordingly until all four IR sensors sense the Gloves again. For example,

let's assume that the sensor only senses the glove on the bottom left IR sensor, then it will assume that the hand is at the bottom left and will rotate in that direction until all four IR sensors sense the glove.

The only issue with this type of system is that the hands' movement mustn't be too quick in order not to risk going outside of the Distance sensors range. However this isn't an issue when performing surgery which is more about small and precise movements.

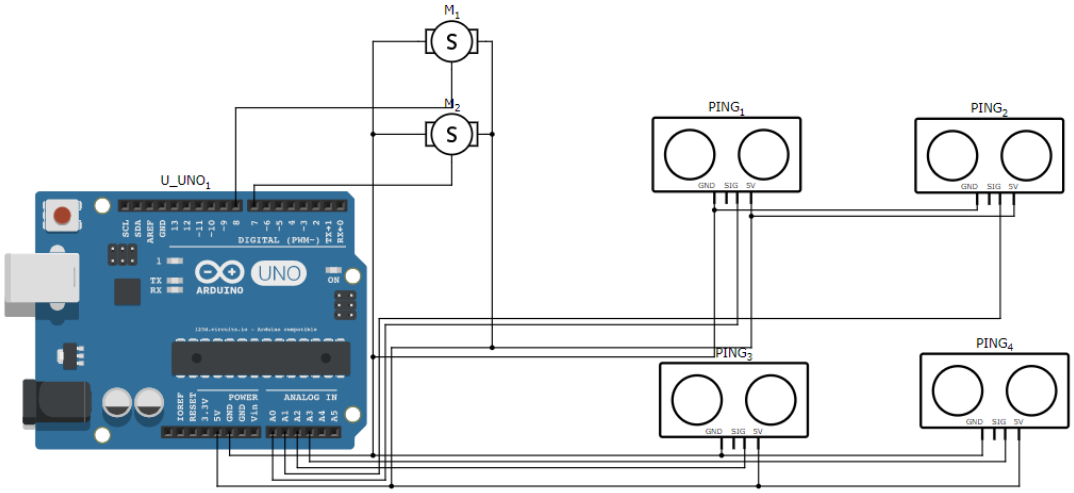


Figure 10: Distance sensor schematics

8.2 Software Implementation

The primary goal of software implementation is to acquire the raw data from the hardware, analyze the data and process it to produce an accurate simulation of the gloves in the virtual reality environment. The hardware, being the Arduino receiving data from the accelerometers and gyroscope, sending the processed data over serial port to the computer where it is once again processed and simulated.

After the data is received from the accelerometers and the gyroscope in the Arduino, they were originally in the void setup function with a data rate of 9600, then followed with multiple conditional if statements to ensure the both accelerometers on the index finger and the thumb along with the palm gyroscope were connected properly. This can be shown here:

Right_Glove

```
#include "SparkFunLIS3DH.h"
#include "SparkFunLSM6DS3.h"
#include "Wire.h"
#include "SPI.h"

LIS3DH SensorOne( I2C_MODE, 0x19 );
LIS3DH SensorTwo( I2C_MODE, 0x18 );
LSM6DS3 Gyro( I2C_MODE, 0x6B );

const int buttonPin = 7;
int buttonState = 0;

void setup() {
// put your setup code here, to run once:
pinMode(buttonPin, INPUT);
  Serial.begin(9600);
  delay(1000); //relax...
  Serial.println("Processor came out of reset.\n");

  //Call .begin() to configure the IMUs
  if( SensorOne.begin() != 0 )
  {
    Serial.println("Problem starting the sensor at 0x19.");
  }
  else
  {
    Serial.println("Sensor at 0x19 started.");
  }
  if( SensorTwo.begin() != 0 )
  {
    Serial.println("Problem starting the sensor at 0x18.");
  }
  else
  {
    Serial.println("Sensor at 0x18 started.");
  }
   if( Gyro.begin() != 0 )
  {
    Serial.println("Problem starting the sensor at 0x6B.");
  }
  else
  {
    Serial.println("Sensor at 0x6B started.");
  }
}
```

Figure 11: Right Glove Arduino setup function

Followed by the setup function, the void loop function was implemented. Here, the original implementation simply printed each accelerometer and gyroscope's XYZ value to the fourth decimal place on a newline. Logically, this makes sense since it will be organized and the values will be segregated. However, it was quickly discovered during testing that the serial values read on Unity only read a single line at a time. Due to this, Unity was only able to read the first line being sent from Arduino over serial port which was the right hand's accelerometer's x-value. The other values were either never received by Unity or Unity was not able to read it properly. The code for the void loop function is shown here:

```
void loop() {
  // put your main code here, to run repeatedly:

  buttonState = digitalRead(buttonPin);
 if (buttonState == LOW){
//Get all parameters

  Serial.print("\nRIX = ");
  Serial.println(SensorOne.readFloatAccelX(), 4);
  Serial.print(" RIY = ");
  Serial.println(SensorOne.readFloatAccelY(), 4);
  Serial.print(" RIZ = ");
  Serial.println(SensorOne.readFloatAccelZ(), 4);
  Serial.print(" RIX = ");
  Serial.println(SensorTwo.readFloatAccelX(), 4);
  Serial.print(" RIY = ");
  Serial.println(SensorTwo.readFloatAccelY(), 4);
  Serial.print(" RIZ = ");
  Serial.println(SensorTwo.readFloatAccelZ(), 4);

 // Serial.print("\nGyroscope:\n");
  Serial.print(" RHX = ");
  Serial.println(Gyro.readFloatGyroX(), 4);
  Serial.print(" RHY = ");
  Serial.println(Gyro.readFloatGyroY(), 4);
  Serial.print(" RHZ = ");
  Serial.println(Gyro.readFloatGyroZ(), 4);

  delay(2);
}

else {
  Serial.println("restart");
  delay(5000);
}
}
```

Figure 12: Right Glove Arduino initial loop function

Along with this serial port limitation of only allowing a single stream of data to be sent over, it was also later discovered through testing that the “readFloatAccel” function produced a value within the range of approximately -2 to +2. This discovery will later be discussed in testing portion of the documentation. However, this range was not ideal since Unity’s 3D virtual reality environment uses values with the 360 degree aspect. Due to these constraints, a “map” function was created, in order to map any value within a range of -2 to +2 to a new range of -180 to +180. These values would represent the degree of rotation for each accelerometer and gyroscope’s XYZ values. With these changes in needed, minor changes were made in the initial “void setup” function. The added code, consisting of new initiated arrays and mapping of each variable, is shown here:

```
const int buttonPin = 7;
int buttonState = 0;

float prevVel[9] = {0};
float prevAcl[9] = {0};
float currentAcl[9];
float currentVel[9];
float currentPos[9];

void setup() {
// put your setup code here, to run once:
pinMode(buttonPin, INPUT);
  Serial.begin(9600);
  delay(1000); //relax...
```

Figure 13: Right Glove Arduino initiated arrays

```
  else
  {
    //Serial.println("Sensor at 0x6B started.");
  }

  prevAcl[0] = map(SensorOne.readFloatAccelX(), -2, 2, -180, 180);
  prevAcl[1] = map(SensorOne.readFloatAccelY(),-2, 2, -180, 180);
  prevAcl[2] = map(SensorOne.readFloatAccelY(),-2, 2, -180, 180);
  prevAcl[3] = map(SensorTwo.readFloatAccelX(),-2, 2, -180, 180);
  prevAcl[4] = map(SensorTwo.readFloatAccelY(),-2, 2, -180, 180);
  prevAcl[5] = map(SensorTwo.readFloatAccelZ(),-2, 2, -180, 180);
  prevAcl[6] = map(SensorThree.readFloatAccelX(),-2, 2, -180, 180);
  prevAcl[7] = map(SensorThree.readFloatAccelY(),-2, 2, -180, 180);
  prevAcl[8] = map(SensorThree.readFloatAccelZ(),-2, 2, -180, 180);

}
```

Figure 14: Right Glove Arduino mapping

In addition to the mapping, due to the fact that the accelerometers gave acceleration values in terms of g-force, they required integration in order to produce position values. This is because the virtual reality simulation takes position values as inputs, and not acceleration values. To work around this problem, Riemann integration was used twice in order to double integrate from acceleration to position values. Once all the XYZ values for both accelerometers and gyroscope are mapped appropriately and stored to their designated array elements, each element is then fed through the initial for loop. This initial for loop acts as the primary integration to integrate from acceleration to velocity. For each element, the average of the previously obtained acceleration is averaged to the most recently obtained acceleration. It is then multiplied by a time value of 1, which would produce the same value which is why it does not show up in the code. The completion of these processes provide the first rectangle for the first Riemann integration calculation to get velocity. Before moving onto the next element, the previous acceleration "prevAcl" value is updated in preparation for the next loop iteration. The same process is then repeated for the second Riemann integration to obtain the position values from the velocity values. The

map function along with its use and the Riemann for loop integration can be seen here:

RightGloveTest3

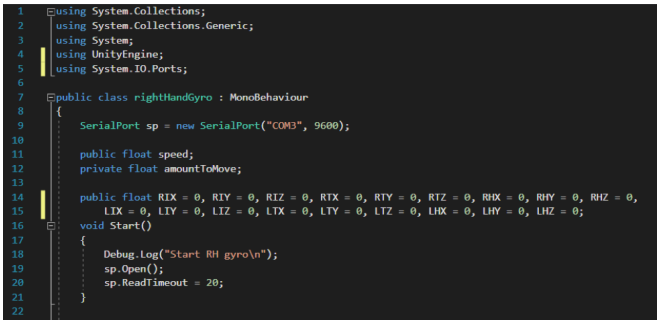
```
void loop() {
  // put your main code here, to run repeatedly:

  buttonState = digitalRead(buttonPin);
 if (buttonState == LOW){
//Get all parameters

  currentAcl[0] = map(SensorOne.readFloatAccelX(), -2, 2, -180, 180);
  currentAcl[1] = map(SensorOne.readFloatAccelY(),-2, 2, -180, 180);
  currentAcl[2] = map(SensorOne.readFloatAccelZ(),-2, 2, -180, 180);
  currentAcl[3] = map(SensorTwo.readFloatAccelX(),-2, 2, -180, 180);
  currentAcl[4] = map(SensorTwo.readFloatAccelY(),-2, 2, -180, 180);
  currentAcl[5] = map(SensorTwo.readFloatAccelZ(),-2, 2, -180, 180);
  currentAcl[6] = map(SensorThree.readFloatAccelX(),-2, 2, -180, 180);
  currentAcl[7] = map(SensorThree.readFloatAccelY(),-2, 2, -180, 180);
  currentAcl[8] = map(SensorThree.readFloatAccelZ(),-2, 2, -180, 180);

  for (int i = 0; i < 9; i++){
    currentVel[i] = currentAcl[i] - ((currentAcl[i] - prevAcl[i])/2);
    prevAcl[i] = currentAcl[i];
  }

  for (int i = 0; i < 9; i++){
    currentPos[i] = currentVel[i] - ((currentVel[i] - prevVel[i])/2);
    prevVel[i] = currentVel[i];
  }

  Serial.print(currentPos[0], 4);
  Serial.print(",");
  Serial.print(currentPos[1], 4);
  Serial.print(",");
  Serial.print(currentPos[2], 4);
  Serial.print(",");
  Serial.print(currentPos[3], 4);
  Serial.print(",");
  Serial.print(currentPos[4], 4);
  Serial.print(",");
  Serial.print(currentPos[5], 4);
  Serial.print(",");
  Serial.print(currentPos[6], 4);
  Serial.print(",");
  Serial.print(currentPos[7], 4);
  Serial.print(",");
  Serial.println(currentPos[8], 4);

  delay(2);
}
```

Figure 15: Right Glove Arduino updated loop function

As shown in the code block, the two Riemann integration for loops are followed by a series of serial prints. The difference with this most recent code block compared to the previous void loop code block is that there is no print newlines in this one. As previously explained, since Unity appears to only be able to read one line per serial transmission, each of the accelerometers' and gyroscope's XYZ values were no longer separated with a new line but now delimited with a comma. While still maintaining a precision of four decimal places, the new concatenated serial transmission can now send all the

data over to Unity on a single line where it is split by a specified delimiter symbol.

Unity is set up with a total of six scripts consisting of two gyroscope scripts for each hand and four accelerometer scripts for both index fingers and thumbs on each hand. When looking at the gyroscope script, it is similar to most standard Unity scripts consisting of the libraries used along with initiated variables and the void start and update methods. The script functionality begins with the void Start method where the debug log and serial port are both opened. The serial port is set to have a ReadTimeout value of 20, equating to 20ms. This is due to Arduino being cable of sending out data faster than Unity is able to receive. With this situation, Unity is set to receive data every 20ms after Arduino produces an average of 10 data points (since it produces values every 2ms). The initial portion of the Unity script can be seen here:



```
using System.Collections;
using System.Collections.Generic;
using System;
using UnityEngine;
using System.IO.Ports;

public class rightHandGyro : MonoBehaviour
{
    SerialPort sp = new SerialPort("COM3", 9600);

    public float speed;
    private float amountToMove;

    public float RIX = 0, RIY = 0, RIZ = 0, RTX = 0, RTY = 0, RTZ = 0, RHX = 0, RHY = 0, RHZ = 0,
        LIX = 0, LIY = 0, LIZ = 0, LTX = 0, LTY = 0, LTZ = 0, LHX = 0, LHY = 0, LHZ = 0;
    void Start()
    {
        Debug.Log("Start RH gyro\n");
        sp.Open();
        sp.ReadTimeout = 20;
    }

```

Figure 16: Initial portion of Unity Script

Followed by the void Start method is the Update method. Within this method, the amountToMove value is first updated relative to the real time vs Time.deltatime and the speed of the object. It is then followed by an if condition to ensure that the serial port is receiving data, then is followed up with a try catch block. Within the try block, the values are first stored into an array of type float through a method created called SerialToFloats. This method will be explained in more detail later, however it is the method which parses through the concatenated stream of data received from Arduino where it is not stored in separate elements within this array. Each array element is then stored into a global variable for public access to other classes in order for them to be inheritable. As shown in the code below, the comments show the initial orientation problems regarding the XYZ values since the gyroscope and accelerometer XYZ did not match Unity's. This will be explained more in-depth in the Testing portion of the report. Followed by setting the global variables to their proper values from the array, they are then set to the game object's rotation value using the built-in Quaternion.Euler function. This function is necessary in order to let the physics engine know that the rotation is relative to the game object it is attached to, and not relative to the virtual reality world. This concludes the rotational functionality of the two gyroscopes for each hand and how each were implemented, with the code shown here:

```
void Update()
{
    amountToMove = speed * Time.deltaTime;

    if (sp.IsOpen)
    {
        try
        {
            float[] gyroVals = SerialToFloats(sp.ReadLine());
            /*
            RIX = gyroVals[0]; // Accel RIX
            RIZ = -gyroVals[1]; // Accel RIY
            RIY = -gyroVals[2]; // Accel RIZ
            RTX = gyroVals[3]; // Accel RTX
            RTZ = -gyroVals[4]; // Accel RTY
            RTY = -gyroVals[5]; // Accel RTZ
            RHZ = -gyroVals[6]; // Gyro RHX
            RHX = -gyroVals[7]; // Gyro RHY
            RHY = -gyroVals[8]; // Gyro RHZ
            */
            RIX = gyroVals[0]; // Accel RIX
            RIY = gyroVals[1]; // Accel RIY
            RIZ = gyroVals[2]; // Accel RIZ
            RTX = gyroVals[3]; // Accel RTX
            RTY = gyroVals[4]; // Accel RTY
            RTZ = gyroVals[5]; // Accel RTZ
            RHX = -gyroVals[6]; // Gyro RHX //This is -RHZ
            RHY = gyroVals[7]; // Gyro RHY //This is RHY
            RHZ = gyroVals[8]; // Gyro RHZ //This is RHX
            //Debug.LogFormat("time: {0}", Time.fixedDeltaTime);
            //Debug.LogFormat("RHX: {0} RHY: {1} RHZ: {2}", RHX, RHY, RHZ);
            // --- ACCELEROMETER IS FLIPPED ---
            // Unity: X = L/R, Z = F/B, Y = U/D
            // Accel: X = L/R, Y = F/B, Z = U/D (Y & Z are flipped)
            // uX = aX, uZ = -aY, uY = -aZ
            // --- GYROSCOPE IS ALSO FLIPPED (acclerometer values on gyroscope) ---
            // Gyro: X = F/B, Y = L/R, Z = U/D (X, Y, & Z are flipped)
            // uX = -gY, uY = -gZ, uZ = -gX
        }
        catch (System.Exception)
        {
        }
    }
    transform.rotation = Quaternion.Euler(RHY ,RHZ, RHX);
    Debug.LogFormat("Linein:{0}", sp.ReadLine());
}
```

Figure 17: Update function for Palm Script

The SerialToFloats method is a vital method which is used to parse through the stream of data sent from Arduino via serial port and store the values into an array. It takes in a single parameter “message” of string type, where it scans for a specified delimiter. In the case of our script, the delimiter it scans for is a comma. The split elements delimited by the comma are now stored into an array of type string called separatedMessage in order to type match. Once this is successfully done, a new array of type float is initiated with the same size as the separatedMessage array. Each element of separatedMessage is then casted as a float and stored into separatedValues. This is because the previously used Quaternioin.Euler function only accepts variables of type float as its parameters and nothing else. The full implementation of the function SerialToFloats can be seen here:

```
float[] SerialToFloats(string message)
{
    string[] separatedMessage = message.Split(',');
    float[] separatedValues = new float[separatedMessage.Length];
    for (int i = 0; i < separatedMessage.Length; i++)
    {
        separatedValues[i] = float.Parse(separatedMessage[i]);
    }
    return separatedValues;
}
```

Figure 18: SerialToFloats function for Palm Script

The classes that inherit values from the palm game object consist of: rightHandIndexInherit, rightHandThumbInherit,

leftHandIndexInherit, and leftHandThumbInherit. These classes mostly consist of a similar class structure, with the differentiating factor being that they inherit different XYZ values. Each script's class begins with a public gameobject class object that stores the game object, and another private variable storing the components that will be used for the game object. There is an void Awake function that is automatically entered right after the virtual reality environment is created and right before any other parts of any script is ran. The debug log ensures that all the awake functions are operating properly, then the game object is located by using GameObject.Find and entering the location of the game object as the address. GetComponent is then used to inherit all the values from the object. All void Start functions for these inherit classes are empty, with the exception of a transform.Rotate line for testing purposes. Finally, all the inherit classes contain an update function that stores the inherited values locally, and the values are then used to rotate the local game object. All four inherit scripts are shown here:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class rightHandIndexInherit : MonoBehaviour {

    public GameObject palm;
    private rightHandGyro righthandgyro;

    public float RIX = 0, RIY = 0, RIZ = 0;

    void Awake () {
        Debug.Log("Awake RH index\n");
        palm = GameObject.Find("Cap1/right hand/Armature/wrist/palm");
        righthandgyro = palm.GetComponent<rightHandGyro>();
        //Debug.Log("Gets RIX val: {0}" + righthandgyro.RIX);
    }

    // Use this for initialization
    void Start () {
        //transform.Rotate(RIX * 0.01f, RIY * 0.01f, RIZ * 0.01f);
    }

    // Update is called once per frame
    void Update () {
        RIX = righthandgyro.RIX;
        RIY = righthandgyro.RIY;
        RIZ = -(righthandgyro.RIZ);
        transform.rotation = Quaternion.Euler(RIX,RIZ,RIY);
    }
}
```

Figure 19: rightHandIndexInherit Script in Unity

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class rightHandThumbInherit : MonoBehaviour {

    public GameObject palm;
    private rightHandGyro righthandgyro;

    public float RTX = 0, RTY = 0, RTZ = 0;

    void Awake () {
        Debug.Log("Awake RH thumb\n");
        palm = GameObject.Find("Cap1/right hand/Armature/wrist/palm");
        righthandgyro = palm.GetComponent<rightHandGyro>();
        //Debug.Log("Gets RIX val: {0}" + righthandgyro.RIX);
    }

    // Use this for initialization
    void Start () {
        //transform.Rotate(RIX * 0.01f, RIY * 0.01f, RIZ * 0.01f);
    }

    // Update is called once per frame
    void Update () {
        RTX = righthandgyro.RTX;
        RTY = righthandgyro.RTY;
        RTZ = -(righthandgyro.RTZ);
        transform.rotation = Quaternion.Euler(RTX, RTZ, RTY);
    }
}
```

Figure 20: rightHandThumbInherit Script in Unity

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class leftHandIndexInherit : MonoBehaviour {

    public GameObject palm;
    private leftHandGyro lighthandgyro;

    public float LIX = 0, LIY = 0, LIZ = 0;

    void Awake () {
        Debug.Log("Awake LH index\n");
        palm = GameObject.Find("Cap1/left hand/Armature/wrist/palm");
        lighthandgyro = palm.GetComponent<leftHandGyro>();
        //Debug.Log("Gets RIX val: {0}" + righthandgyro.RIX);
    }

    // Use this for initialization
    void Start () {
        //transform.Rotate(RIX * 0.01f, RIY * 0.01f, RIZ * 0.01f);
    }

    // Update is called once per frame
    void Update () {
        LIX = lighthandgyro.LIX;
        LIY = lighthandgyro.LIY;
        LIZ = lighthandgyro.LIZ;
        transform.rotation = Quaternion.Euler(LIX, LIY, LIZ);
    }
}
```

Figure 21: leftHandIndexInherit Script in Unity

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class leftHandThumbInherit : MonoBehaviour {

    public GameObject palm;
    private leftHandGyro lighthandgyro;

    public float LTX = 0, LTY = 0, LTZ = 0;

    void Awake(){
        Debug.Log("Awake LH thumb\n");
        palm = GameObject.Find("Cap1/left hand/Armature/wrist/palm");
        lighthandgyro = palm.GetComponent<leftHandGyro>();
        //Debug.Log("Gets RIX val: {0}" + righthandgyro.RIX);
    }

    // Use this for initialization
    void Start(){
        //transform.Rotate(RIX * 0.01f, RIY * 0.01f, RIZ * 0.01f);
    }

    // Update is called once per frame
    void Update(){
        LTX = lighthandgyro.LIX;
        LTY = lighthandgyro.LIY;
        LTZ = lighthandgyro.LIZ;
        transform.rotation = Quaternion.Euler(LTX, LTY, LTZ);
    }
}
```

Figure 22: leftHandThumbInherit Script in Unity

9 Testing and Evaluation

9.1 Hardware Testing/Evaluation

Due to the nature of the project, a lot of testing was done throughout the implementation of the project. The testing of hardware and software components of the project will be explained further in this section.
The testing of the voltage and current of the circuit was done with the multi-meter. There was multiple test done in order to make sure that the circuit will continue to work as more components are added to this integrated project. The oscilloscope will be used to test the signal output when we assemble the VRSS and test if we could get the correct wave function as the output to ensure that the circuit was functioning properly.

Multiple tests were done over the course of this project. This is because to avoid any potential disasters and add more costs towards the budget. Tests that would had to be done includes the following: voltage, current, motors, camera focusing, sensor testing, as well as make sure it works properly with our code in C# language. The simulation testing can be done with Matlab, whenever we will have to re-calibrate the system. This also prevented any unwanted consequences.

The IR sensor testing was done manually. Each IR sensor was placed such that it measures distance horizontally parallel to the ground. With a ruler and a piece of cardboard, values were taken for each centimeter away from the sensor as seen in table 5. From these values, a graph (figure 9) was drawn in excel in order to determine the minimum threshold for the

sensor not to get confused.

9.2 Software Testing and Evaluation

Due to the nature of the project, a lot of the testing was done throughout the implementation of the project. In terms of testing for the software portion, a lot of problems were discovered early on in development. As explained in implementation, one of the main problems with the use of serial port for sending data from Arduino to Unity was the fact that only one line of data can be received at a time. Many iterations of code were return in order to test for this. Initially, the code was written to simply send a simple count value every second as shown here:

serialTesting

```
int count = 0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  count = count + 1;
  if(count == 10000)
    count = 0;
  Serial.print("Teste: ");
  Serial.println(count);
  delay(1000);
}
```

Figure 23: Initial Arduino Testing

This ran without a problem, making it appear as though there were no problems with print line. However, this was due to each serial print only printing one line therefore not causing any multi-line print problems. This code was further modified using multiple count variables and multiple prints to verify the nature of Unity receiving serial lines since the original design process was to print every XYZ value of the accelerometers and gyroscopes on new lines. This testing procedure was later modified to use multiple count variables in order to verify the number of variables Unity was able to receive and which lines it is able to read. This code can be seen below, where certain lines were commented or uncommented during testing, depending on the results received.

serialTest2 §

```
int count = 1;
int count2 = 0;

void setup() {
  Serial.begin(9600);
  count2 = count2 + 1;
  Serial.print("count is now ");
  Serial.print(count);
}

void loop() {
 Serial.print(", Count2: ");
 Serial.println(count2);

  // --- COUNT 1 ---
  if(count >= 0)
  {
   //Serial.println("count is >= 0");
   count = count + 1;
   //Serial.println("count + 1");
   //Serial.print("Count VALUE is now");
   //Serial.println(count);
   if(count == 3)
   {
     count = -1;
     //Serial.println("count is =3, count is now -1");
   }
  }
  if(count < 0)
  {
   count = count -1;
   //Serial.println("count is < 0, count -1");
   if(count == -5)
   {
     count = 0;
  //   Serial.println("count is -3, count is now 0");
   }
  }

  // --- COUNT 2 ---
  if(count2 >= 0)
   count2 = count2 + 1;

  Serial.print("Test: ");
  Serial.println(count);
  delay(1000);
}
```

Figure 24: Arduino Testing with Count Variables

Once it was discovered that Unity was unable to receive or read multiple serial print lines, the current implementation was used as shown in the Implementation portion this documentation. The final major problem associated with Arduino, however, was discovered during the initial testing of the movement of the hands in virtual reality. The rotational values in

Unity are set in terms of rotational degrees, while the gyroscopes and accelerometers had somewhat arbitrary values. Through testing in the Arduino serial monitor, it was discovered that the gyroscopes and accelerometers had around a -2 to +2 range for every 180 degrees turned in a negative and positive direction. The reasoning for testing from negative 180 degrees to positive 180 degrees is because rotating both directions at 180 degrees encompasses every possible turn in a complete 360 degree rotation. Based on these results, the raw -2 to +2 values generated from the gyroscopes and accelerometers were mapped to a -180 to +180 range in Arduino with function created called map.

The function contains five parameters x representing the value that will be mapped, in(min) and in(max) representing the initial min and max that the x value will be contained within, and finally the out(min) and out(max) representing the output min and max that the x value will be mapped to. With the values for all the XYZ values for all accelerometers and gyroscopes, they were able to get concatenated to be sent via serial port over to Unity.

When receiving values on Unity, since each finger acts independently from one another, they were each required to be independent game objects with their own independent scripts. Initially, each script was set to receive all the values from Arduino, as shown in the "rightHandGyro" script in the Implementation portion of the document. However, this quickly caused malfunctions since each game object (i.e. each finger and moving game object) was attempting to open a serial port simultaneously on the same COM port. This caused communication issues for the game objects conflicting each other on the right hand and the game objects conflicting each other on the left hand. This issue was solved by making the palm game object, which uses the gyroscope's XYZ values, the parent object to all the fingers. By doing so, the finger game objects were able to access all public variables and functions from the palm game object. This allowed the palm game object to receive all the XYZ values via serial port while each finger simply inherited the values they required.

Finally, the last major testing portion involves the actual precision of the simulation of the glove movements. Initially, the XYZ values were simply mapped directly onto each game object's rotation coordinates. The problem was evident right on the first simulation, since there were many things rotating in the wrong direction. The problem was quickly discovered that the XYZ vectors for the gyroscopes and accelerometers did not match the XYZ vectors. The first reasoning that was discovered to cause the problem was due to the accelerometers being installed upside down. The reasoning for this is due to the pins on the accelerometer needing to stick up and not down, or else they would stab the user's finger. After consulting the accelerometer documentation, it was found that the accelerometer's X dimension emulated left and right, Y dimension emulated front and back, Z dimension emulated up and down. Meanwhile, Unity's X dimension was the same as the

accelerometer's native X dimension which simulated left and right, while the Y and Z dimensions were flipped from the accelerometer (i.e. in Unity the Y dimension simulated up and down and the Z dimension simulated front and back). The initial approach was to just map Unity's XYZ vectors to the accelerometer's and gyroscope's native XYZ, however minor problems persisted. This led to the final extensive set of tests, where each individual dimension's direction for each accelerometer and gyroscope was testing. This was done by setting the other dimensions to zero while testing the current dimension (i.e. (X,Y,Z) was set to (X,0,0) to test for the X dimension's direction). The initial mapping for this can be shown below:

```
try
{
    float[] gyroVals = SerialToFloats(sp.ReadLine());
    /*
    RIX = gyroVals[0]; // Accel RIX
    RIZ = -gyroVals[1]; // Accel RIY
    RIY = -gyroVals[2]; // Accel RIZ
    RTX = gyroVals[3]; // Accel RTX
    RTZ = -gyroVals[4]; // Accel RTY
    RTY = -gyroVals[5]; // Accel RTZ
    RHZ = -gyroVals[6]; // Gyro RHX
    RHX = -gyroVals[7]; // Gyro RHY
    RHY = -gyroVals[8]; // Gyro RHZ
    */
    RIX = gyroVals[0]; // Accel RIX
    RIY = gyroVals[1]; // Accel RIY
    RIZ = gyroVals[2]; // Accel RIZ
    RTX = gyroVals[3]; // Accel RTX
    RTY = gyroVals[4]; // Accel RTY
    RTZ = gyroVals[5]; // Accel RTZ
    RHX = -gyroVals[6]; // Gyro RHX //This is -RHZ
    RHY = gyroVals[7]; // Gyro RHY //This is RHY
    RHZ = gyroVals[8]; // Gyro RHZ //This is RHX
    //Debug.LogFormat("time: {0}", Time.fixedDeltaTime);
    //Debug.LogFormat("RHX: {0} RHY: {1} RHZ: {2}", RHX, RHY, RHZ);
    // --- ACCELEROMETER IS FLIPPED ---
    // Unity: X = L/R, Z = F/B, Y = U/D
    // Accel: X = L/R, Y = F/B, Z = U/D (Y & Z are flipped)
    // uX = aX, uZ = -aY, uY = -aZ
    // --- GYROSCOPE IS ALSO FLIPPED (acclerometer values on gyroscope) ---
    // Gyro: X = F/B, Y = L/R, Z = U/D (X, Y, & Z are flipped)
    // uX = -gY, uY = -gZ, uZ = -gX
}
```

Figure 25: Mapping Dimensions in Unity

While comparing the two testing methods, it was decided that although the second testing method required a lot more time, it was the more precise testing method that provided the better results. Although the initial testing method of mapping the XYZ directions based on documentation should technically work flawlessly, there are many calibration problems that can occur on such a custom set up like the VRSS. The latter method allowed the team to calibrate the gloves specifically for this system, allowing for the most accurate simulation possible with the given setup.

The major standard for this project would be to try to create a working prototype of the VRSS surgical simulation system. Performance standard would include have to be: to have a positioning error of +/- 2 cm, to have a minimum lag response in the feedback system, to have accurate calibration, in other words to be able to accurately analyze and check the users

movements every second.An example of feedback system can be seen in the figure below. Due to complications within the re- calibration of the hands in the virtual environment as well as the problem with the triple axis accelerator on the surgical gloves, the previous objection standard for having the positioning error changed to compensate for the error in lag time in our simulation.

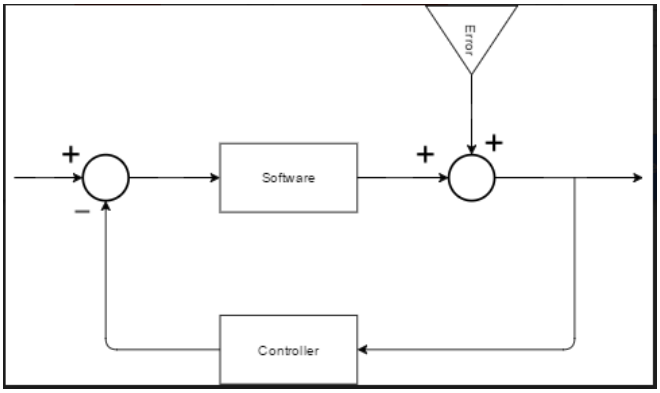


Figure 26: Example Feedback loop for system lag

10 Conclusion

The VRSS, which stands for the virtual reality surgery simulator is a simulator that would help surgeons and medical students alike in their procedures. The main point of the VRSS is to be used by medical students as a training simulation during their first 2 years of medical school that are mostly spent in the classroom. The VRSS would give these students a first person view of the surgeries that they would be performing in the future, as well as a feel of the atmosphere during a surgery. They would be able to use this simulation at any time and without wasting resources. The VRSS could also be used by surgeons to model special cases and allow the surgeons to prepare for upcoming procedure.

The three main components of our project are a distance sensor, a pair of gloves that measure the movement of the hands and a VR simulation that incorporate the measurements of the other two components.

The next steps for the VRSS project would include the following: implementing the PCB board schematics to make the depth sensor and the surgical gloves less robust and more aesthetically pleasing as well as making the overall hardware design lighter. Making the gloves wireless by using Bluetooth and rechargeable batteries will also allow the users more freedom of movement.

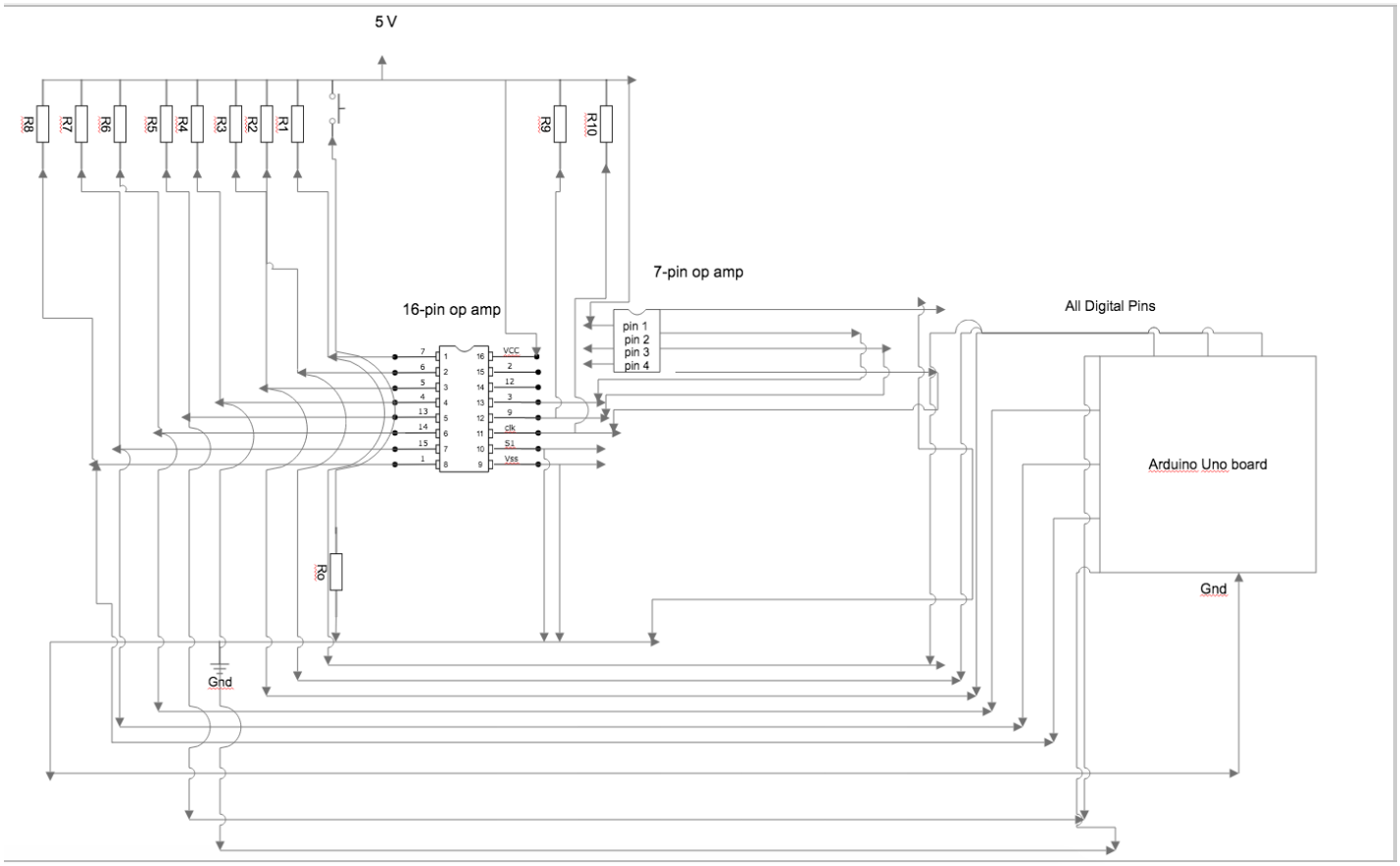
Moving forward, the main software improvements that can be made if the VRSS were to be commercialized would definitely be to include the use of bluetooth. By using bluetooth, not only would the connection be more reliable overall, but it would eliminate the dependence on having multiple larger connections (i.e. using LAN and serial port). This would allow the user the freedom of movement by removing the necessity of relying on a USB connection.

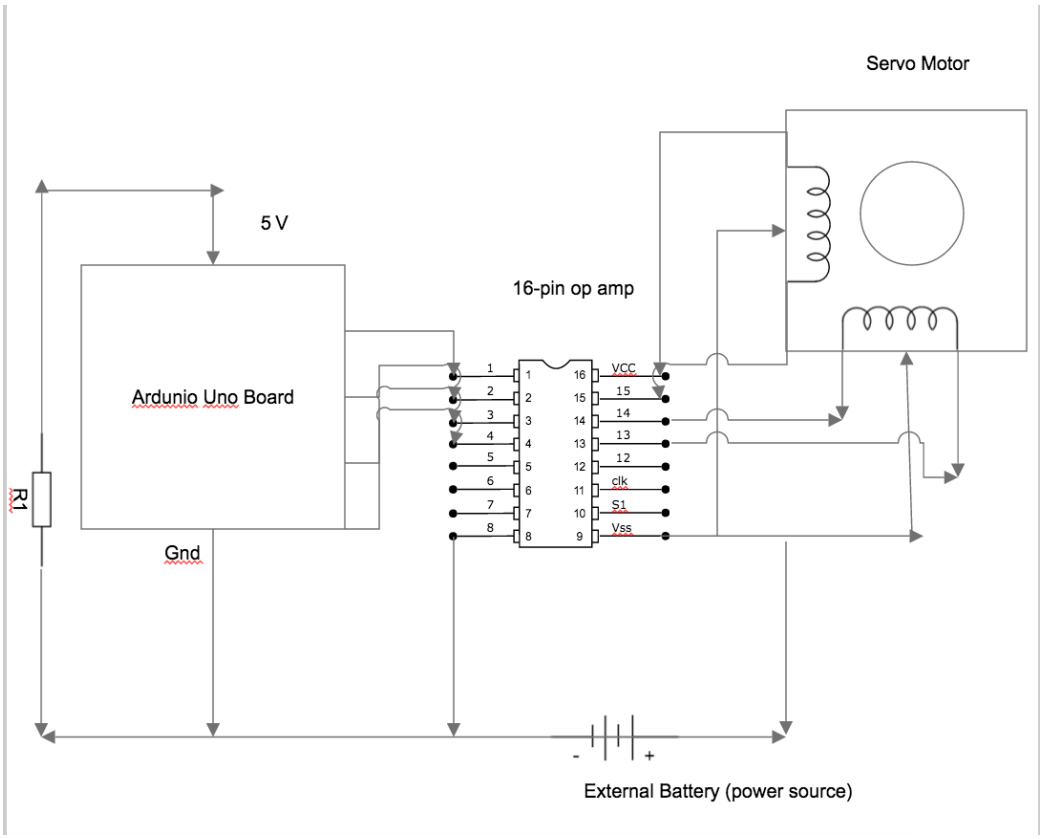
VR is a growing market, especially within the gaming scene, however with more VR projects focusing on real life implementations, large scale commercialization of our project could be viable. The market for the VRSS wouldn't only be focused towards surgeons in training but towards the entire medical field. Furthermore, most of the VRSS's components are already available to the public, with Android phones being used by around 80% of phone users (6) and the only components that need to be purchased are the hardware devices of our project.

The potential impacts of the project as mentioned before are the environmental issues of the overall waste use for the surgical gloves as well as the depth sensor. This can be mitigated by a more effective design with less robust hardware and have sensors built into the surgical gloves itself. As for the distance senor to make the body out of more reusable material. The social impacts include how it affects the training of the medical students in the future. This prove of concept can be the next leading way how to train doctors and this concept can be applied to gaming design and to other fields as well. Legal impacts would be the distribution of the software code and hardware to Queen's University Electrical and Computer Department.

11 Appendix

11.1 Appendix A




Servo Motor
5 V
16-pin op amp
Ardunio Uno Board
R1
Gnd
VCC
15
14
13
12
clk
S1
Vss
External Battery (power source)

11.2 Appendix B

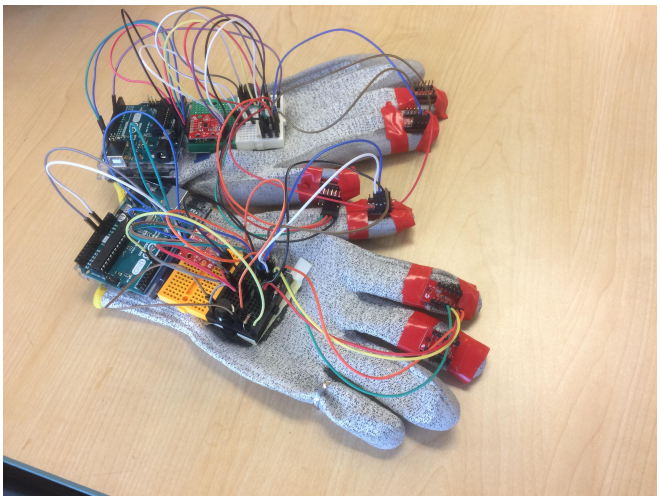


Figure 27: Gloves for the Project

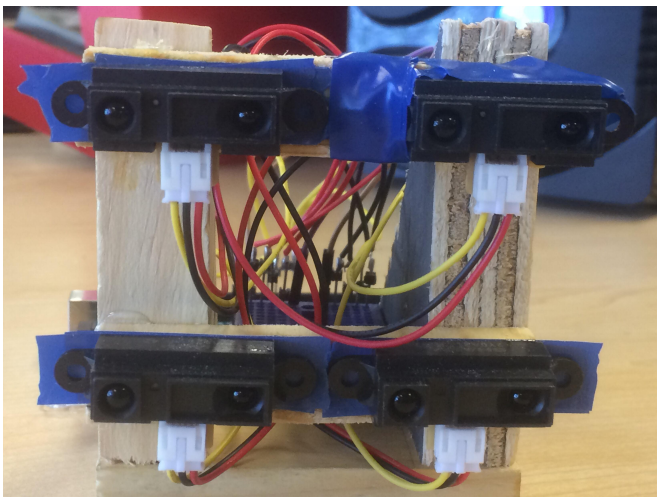


Figure 28: Flow Chart for Proposed Project

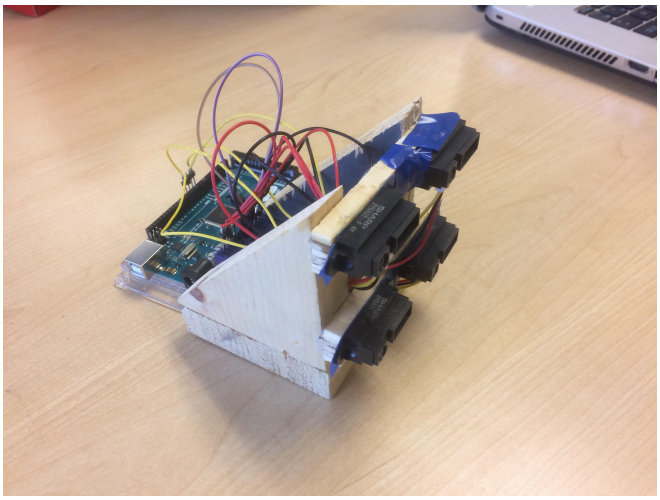


Figure 29: Flow Chart for Proposed Project

11.3 Appendix C

Material	Price	# needed	Total (CA$)	Source
Arduino Uno R3	24.95	4	100	https://www.sparkfun.com/products/11021
Battery Pack	4.95	2	10	https://www.sparkfun.com/products/11222
Accelerometer	4.95	6	30	https://www.sparkfun.com/products/13963
Gyroscope	9.95	2	20	https://www.sparkfun.com/products/13339
Bluetooth	4.26	2	8.52	https://www.ebay.ca/sch/i.html?_from=R40&_trksid=p2380057.m570.l1311.R5.TR8.TRC1.A0.H0.Xhc06.TRS0&_nkw=hc-06+wireless+bluetooth&_sacat=0
Gloves	12.43	1	12.43	https://www.amazon.ca/Skytree-Gloves-Gardening-Comfort-Breathable/dp/B06XYLBGGH/ref=sr_1_1?ie=UTF8&qid=1505922425&sr=8-1&keywords=gardening+gloves+men
Electrical Tape	14.27	1	14.27	https://www.amazon.ca/Scotch-Electrical-Tape-Value-Pack/dp/B001B19FDK/ref=sr_1_3?ie=UTF8&qid=1505922705&sr=8-3&keywords=electrical+tape
CCD camera	14.99	1	14.99	https://www.amazon.ca/XCSOURCE-700TVL-Camera-Photography-AH078/dp/B0157N4QPO/ref=sr_1_5?ie=UTF8&qid=1505847627&sr=8-5&keywords=ccd+camera

Table 6: Blueprint budget

IR laser	55.41 (US)	1	55.41 (US)	https://www.amazon.com/G-Sight-LSBG-IR-Gladiator-Infrared-Laser/dp/B01H5T7PXI/ref=sr_1_1?ie=UTF8&qid=1505847374&sr=8-1&keywords=IR+laser#Ask
Laser beam splitter	11.99 (US)	1	11.99 (US)	http://www.dragonlasers.com/Laser-Beam-Splitters.html
CMOS Camera	31.95	1	31.95	https://www.sparkfun.com/products/11745
Servo Motors				Not sure about supplier or price
PCB Board				Not sure about supplier or price

Figure 30: Blueprint budget

11.4 Appendix D

Adel and Monica Meeting Notes

Thursday, January 4, 2018 1:10 PM

Names: Brain storming ->
"Surgical VR simulator"

-> Adel will get code for distance sensor from 299 group

Budget:
What we have->
-Eric is donating his VR headset (including Android Phone Monica's or (ric's)
-gloves (gardening or work gloves)- Easy purchase
-software
-Monitor (Bain)
-coffee
-electrical kits x2
-volt/amp meter
-Tape (medical tape, Duct tape, Electrical tape etc)
-solder
-soldering iron

NEED TO TALK TO GAZOR-> Budget, PCB board schematics, Blueprint report

What we need to buy:

- Glove sensors (**Motion sensors**) -> need enough for two gloves

4x	**mini Arduino**
2x	**Servo motors (tilt, rotate)**
4x	**16-pin op amps**
2x	**7-pin op amps**

- Varies resistors (few packs, don't know exactly what resistances we need)
- **NEED TO ORDER AND DESIGN PCB BOARDS---> ????**
- **Wires of varies lengths**
- **Sensors (distance sensors, gyroscope sensor, force sensor, infrared sensor, depth sensor)**
- **3-D print shell of Kinect and scalpel -> need to figure this out**

6x	**specialized cameras (RGB(red, green, blue) and Infrared emitter red (IR) and depth camera)**
2x	**Battery packs**
2x	**Bluetooth**
1x	**USB cable**
x	**Capacitors, inductors, resistors,**

RGB Camera: with
https://www.sparkfun.com/products/retired/8667

Kinect workings:
https://www.quora.com/How-does-Microsofts-Kinect-work-from-a-technology-standpoint

Estimate of Cost so far:
- Arduinos **@$25.00 each**
https://www.sparkfun.com/arduino_guide

- Cameras-> IR + depth, RGB, **~ $200**
- Servo motors **~$100**
- Op amp **~$20.00**
- Bluetooth **~$33.00**
- **ELECTRICAL TAPE**
- Battery packs **~$10.00**

https://www.sparkfun.com/products/11222

Gloves:
1- **Motion Sensor**: http://www.glynstore.com/signalquest-sq-min-200-omnidirectional-tilt-vibration-sensor/
https://www.sparkfun.com/products/13963

2- **Gyro**: https://www.sparkfun.com/products/13339

3- **Bluetooth**: https://www.sparkfun.com/products/9434
or: https://www.ebay.ca/sch/i.html?_from=R40&_trksid=p2380057.m570.l1311.R5.TR8.TRC1.A0.H0.Xhc06.TR50&_nkw=hc-06+wireless+bluetooth&_sacat=0

4- **Gloves**: https://www.amazon.com/Custom-Leathercraft-125L-Resistant-Stretchable/dp/B0002YPZLI/ref=sr_1_5?ie=UTF8&qid=1505835060&sr=8-5&keywords=Gardening+gloves+men

5- **Electrical Tape**: https://www.amazon.com/3M-Scotch-Electrical-Value-1045/NA/dp/B00VVN070M/ref=sr_1_6?s=hi&ie=UTF8&qid=1505835410&sr=1-6&keywords=electrical%2Btape&th=1

SO DOING THE MATH WITH THE LINKS, FROM VARIES SITES...
Cost breakdown of surgical gloves

50	(ARDUINO)
10	(BATTERY PACKS)
30	(ACCELEROMETER)
20	(GYROSCOPE)
8.52	(BLUETOOTH) (EBAY PRICING)
12.43	(GLOVES) (AMAZON)
14.62	(ELECTRICAL TAPE) (AMAZON)

$145.05 (not including taxes/shipping)

Cost breakdown of "Kinect"

50	(ARDUNIO)
14.99	**CCD camera**
55.41 (US)	**IR laser**
14.70	**Laser beam splitters**
31.95	**CMOS camera**
	Servo motors

Total cost: $336.64 (not including taxes and shipping)

Sensor:

490 Group Meeting Notes

Thursday, January 4, 2018 1:09 PM

Team #33Meetings FYI:

Blue is team meeting times
Yellow is Adel and Monica
Pink is what is due
Green is what to talk to Gazor
TA Meeting Time

- Deciding on where and when to meet:
- For now Group Meetings Tuesdays : 9:30 am -11:20 am
- When labs start Thursdays 5:30 pm-7:30 pm

-Make a glove (cost?, accuracy, material)
- Adel and Monica meeting talk about budget and glove->
- **Now at Wednesdays from 11:30-1:30 pm**

Meet with Gazor every second week, (ask about assignment)
What time he is free, meet next week,

- **Need to work on Blueprint report due Thursday Sept 21 At 11:59 pm**

Copy some parts from the sharelatex file of 490 proposal
Start a group word document

-Need to find a time to meet with Gazor

NEED TO TALK TO GAZOR->
- **Budget** (how much money do we have to work with)
- PCB board schematics
- Blueprint report

-Need to start and probably build two gloves
-Need to discuss how much are we willing to put in
-Lockers!!
-Breakdown of money where it is going -> Hardware

Adel and Monica meeting for all day on Tuesday!

Monica email TA

Talk about Blueprint report

- Parts are came in, in progress in building gloves, eric shit
- We buy that app, porbs 20 bucks
- Other parts coming in
- Op amps
- Pcbs doing after

Things that need to be ordered need to get done by this week:
New team meeting day Wednesdays and Fridays
Ta meeting day : Fridays at 2:30 pm
Supervisor meeting TBA
Adel and Monica meeting Thursday some time other wise after Taing
Another team meeting day Fridays

Milestones to get done by this week1

Adel: Bluetooth also figure out multiple accelerometer

Monica: solid works/solid edge schematic done for Sunday also look into old camera code

Both we need to get shit together and make one last order form

Eric: half the environment done.

Week 2 Milestones:

Adel: start building gloves

Monica: Build gloves and soldering

Eric: working on environment

Week 3 Milestones:

Adel: (Friday, building gloves Friday), switching mechanism, building

Monica: look at camera, Soldering , (Friday, building gloves Friday), search ->unity ,plastic for distance sensor

Eric: Corps for the environment, working on unity,

Week 4 Milestones:

11.5 Appendix E

1 Pin description

Figure 1. Pin connections

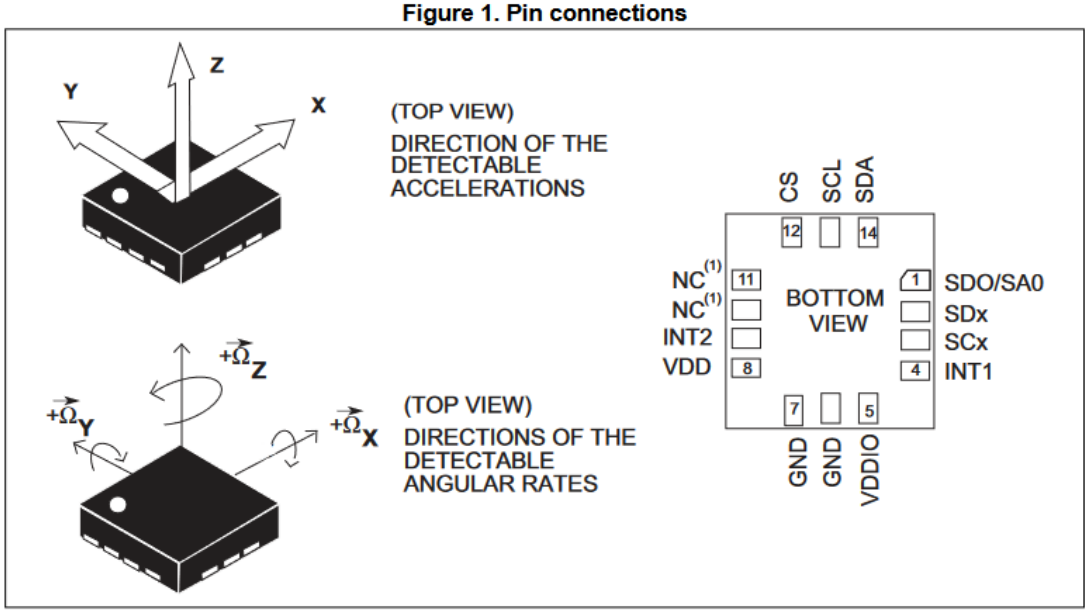


Figure 31: Gyroscope Native Dimensions

1.2 Pin description

Figure 2. Pin connections

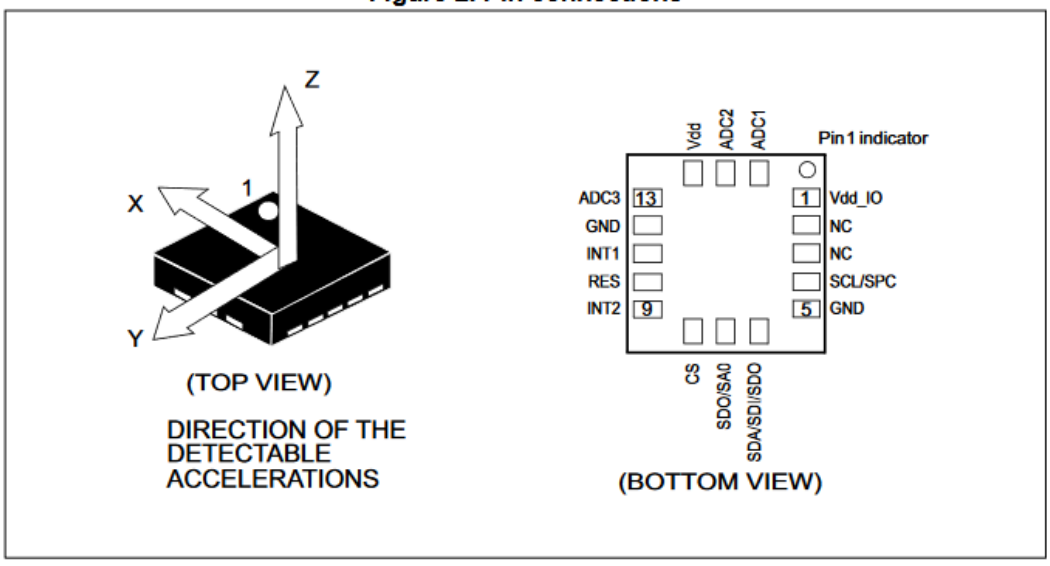


Figure 32: Accelerometer Native Dimensions

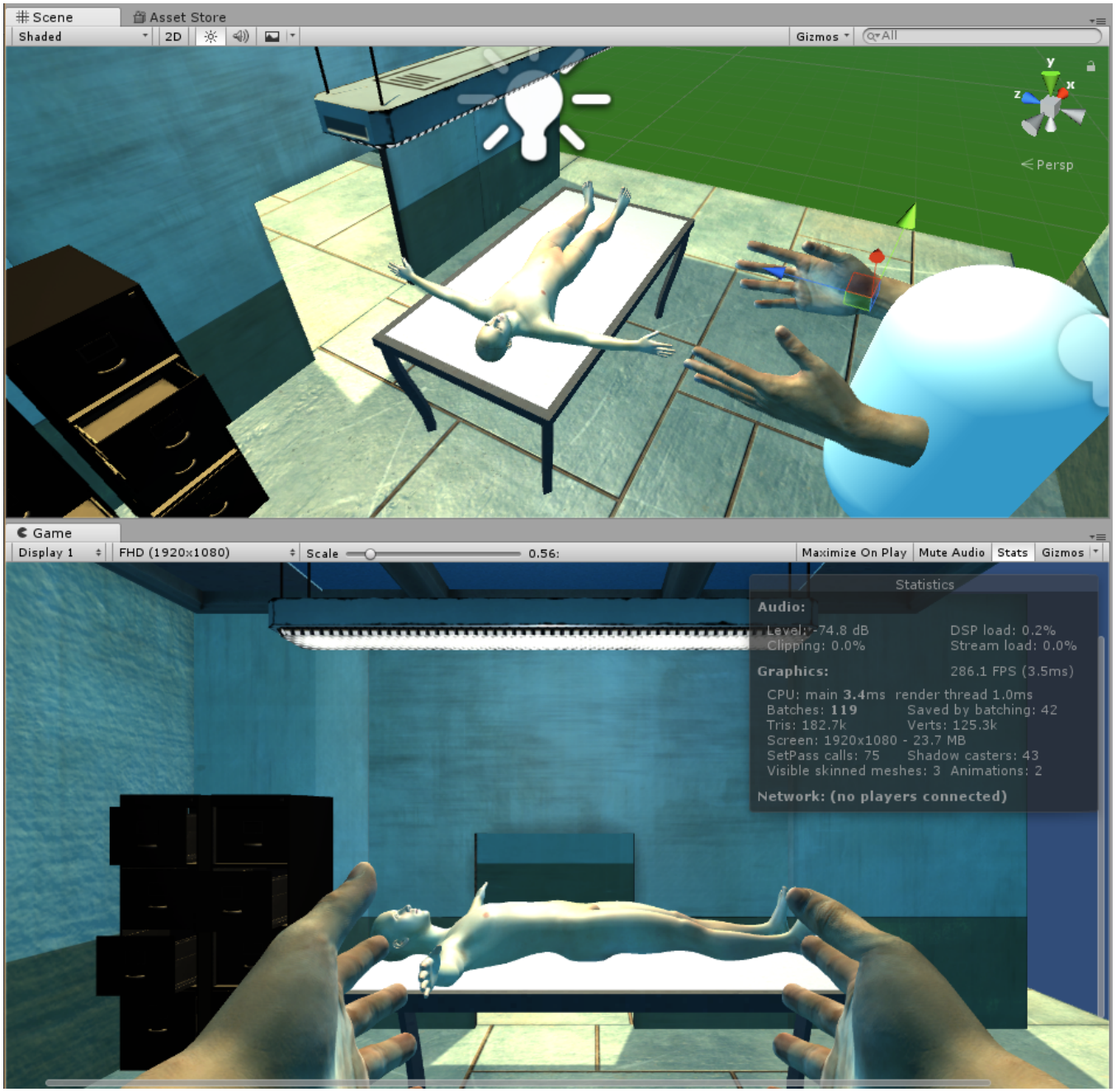


Figure 33: Unity Simulation Example with Unity Native Dimensions

12 Bibliography

References

[1] "The future of surgery: Less cutting, more robots."

[2] "The future of surgery is robotic, data-driven, and artificially intelligent."

[3] "Surgical simulation training: Is virtual reality the future of surgical training?"

[4] "Microsoft kinect-based technology for vascular surgery now being extended to neurosurgery."

[5] "Depth measurement based on infrared coded structured light."

[6] "99.6 percent of new smartphones run android or ios."