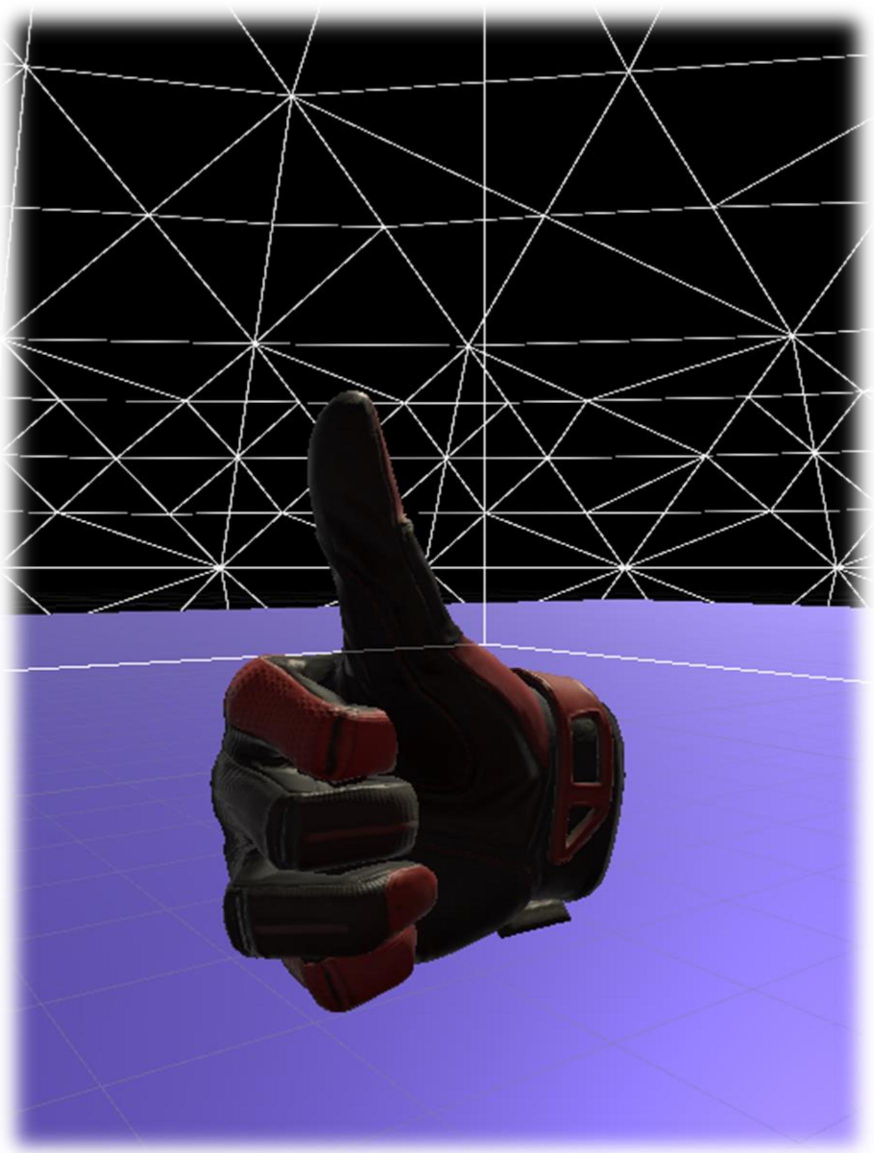
# The Ultimate Controller

Zakhar Kanyuka | CS 390 AP



## Introduction and Justification

### The Human Hand

Few anatomical features give humans as many ways of interacting with their surroundings as their hands. Thanks to the dexterity facilitated by the combination of complex structure, and intricate neurological control over the hand, we have been able to undertake a wide selection of tasks that led us

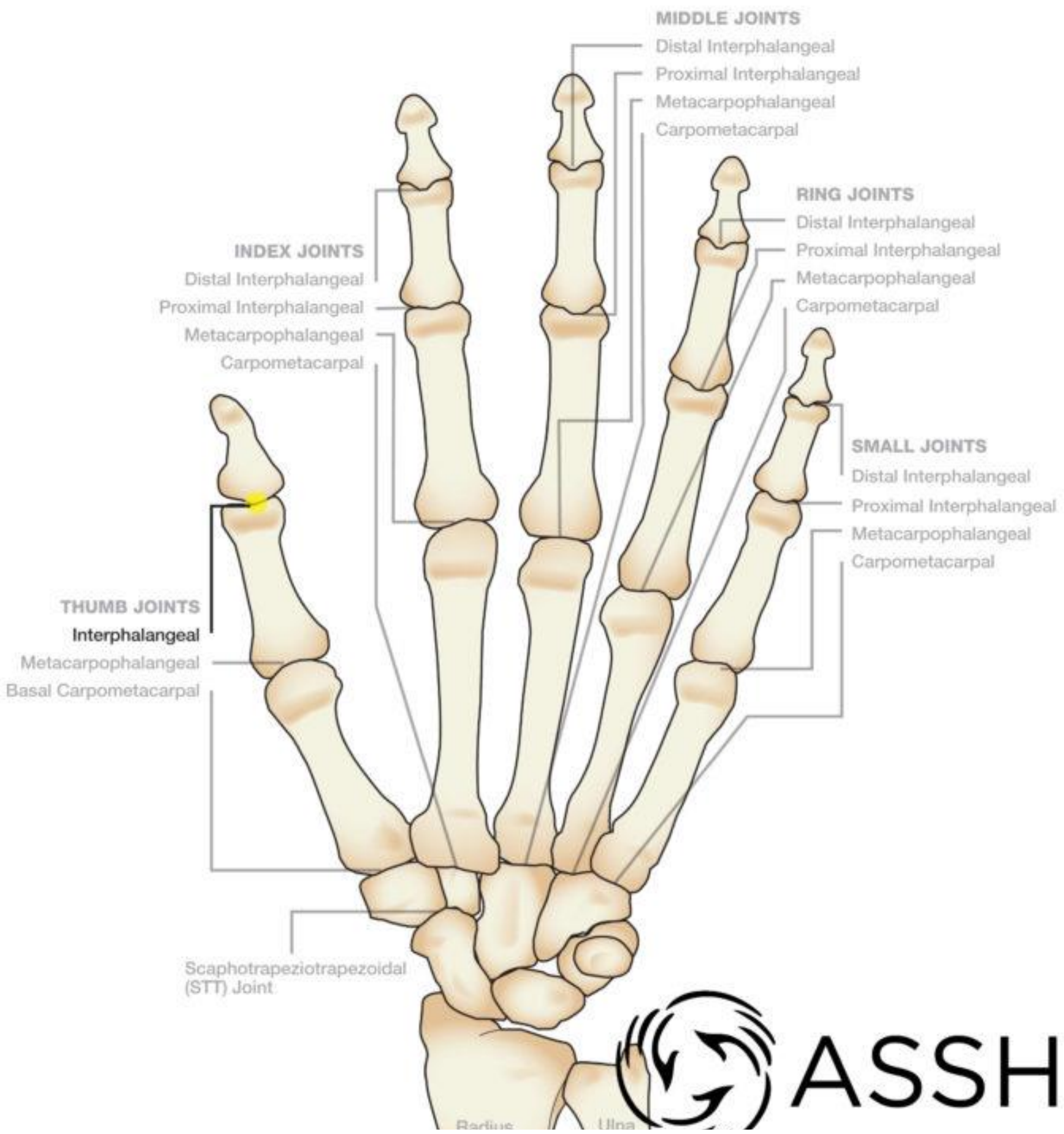


*Figure 1 Anatomy of the hand. Taken from American Society of Surgery Hand Care Blog.*

to the rich, diverse thriving societies we have today. Throughout the evolutionary journey, hands have been key to making and using tools and weapons, growing and harvesting food, creating art, playing games, and sharing ideas. As the staple of the human experience, therefore, it should be inevitable that hands get a worthy representation in virtual reality (VR).

## The Problem with Common Controllers

Ivan Sutherland's "The Ultimate Display," a document viewed by some as the manifesto for the development of VR in years to come, describes the ideal interface as multisensory (Sutherland, 1965). Among other senses, he encourages the engagement of haptics through input mechanisms that convey to the user the feeling of force; although he proposes the use of body parts other than hands and arms to interact with the virtual world, he acknowledges the value of manual interaction due to the hands' superior dexterity (Sutherland, 1965). With the spread of personal computing, most systems take advantage of the users' manual dexterity for their primary sources of input such as keyboards, mouses, pen tablets, and gamepads. Most of these devices also provide rudimentary haptic feedback either through tactile buttons whose engagement the user can easily distinguish, or though vibration motors. While they fall short of taking advantage of numerous degrees of freedom and immense sensory capacity offered by the hands, they are sufficient for navigating the simple, abstract interfaces we see on most two-dimensional displays; this is in part thanks to the agreement among the users and developers of the given systems on standardized control schemes such as pressing 'W,' 'A,' 'S,' and 'D' on a keyboard to move an avatar in a computer game or turning a dial to adjust the shutter speed on a camera. Modern head-based VR systems can transport the user to a world where he or she can naturally interact with three-dimensional objects, no longer limited to the abstract two-dimensional interfaces of pancake systems. Although it can improve the user-friendliness of a system, reaching an agreement on control schemes to be used across VR applications would restrict the possibilities of a virtual universe where nearly anything is possible; using control schemes established for use in a pancake system would hinder the user's ability to naturally manipulate the virtual world. To achieve optimal interaction with a VR experience designed to mechanistically mimic the real world, the peripherals must therefore allow for natural interaction mechanics. An interface which snugly fits the user’s hand such as a glove or an exoskeleton is essential to achieving natural interaction with the virtual world.

## Early Glove-Shaped Peripherals

Sayre Glove, developed by University of Illinois’ researchers in 1977, is credited as the first of glove-based peripherals to come. However, the idea saw little commercial success until the release of the DataGlove by VPL Research in 1987. Taking advantage of the optical flex sensor patent given to them two years prior, and magnetical motion tracking offered by Polhemus, the DataGlove was ground-breaking for the time period. Owing in part to VPL Research founder Jaron Lanier, who coined the term “virtual reality,” the DataGlove is regarded as one of the first peripherals made for VR. While the $8800 asking price kept it of the public’s hands, the DataGlove found its way into research and industrial applications, most notably NASA’s VIEW program aimed at using VR technology for training and telepresence (Pollack, 1989; Rosson, 2014). With the aim of putting a glove-shaped peripheral on the hands of the everyman gamer, Mattel released the inexpensive, $75 Power Glove controller for the Nintendo NES in 1989 (Struman & Zeltzer, 1994). Due to the limited, 2 bits per finger tracking made possible by contemporary technology at the price point and the lack of games available for the system, the Power Glove was a commercial failure (Struman & Zeltzer, 1994). Furthermore, the Power Glove was left out of the field of head-based VR as it was discontinued in 1990, 5 years before Nintendo released the Virtual Boy head-mounted display system. Glove-shaped peripherals soon intersected with the gaming world again when W Industries (precursor to Virtuality) released the Space Glove, technologically intermediate between the DataGlove and the Power glove (Struman & Zeltzer, 1994). The Space Glove did not become widespread among consumers as the W Industries’ VR systems were marketed and priced primarily for arcade use.

## Current State of VR Peripherals

Like Nintendo's Virtual Boy and other low-cost VR systems from the 1990s, many modern head-based VR systems use control schemes carried over from pancake systems. Google's Daydream VR and Samsung's Gear VR used simple remotes with rotational tracking, whereas early Oculus Rift headsets shipped with Xbox One gamepads. To make better use of the spatial tracking of room-scale VR, Oculus's Rift and HTC's Vive systems received controllers that shared the precise positional and rotational tracking technologies of their respective headsets. Combined with intuitive button layouts, this gave the user a less restricted interaction with the virtual world than ever, granting the ability to approach, pick up, throw and point objects in any direction. However, even when using the innovative controllers, the users remain largely tied to the joysticks, buttons, and triggers carried over from gamepads of the past. This comes back to the incompatibility of pancake system control schemes for VR applications. For example, although the user can use the controller's motion tracking to place his or her hand over an interactable element in the virtual world such as a drawer handle, grabbing at it remains puzzling – should one press a specific button on the controller, squeeze the hand grip, or merely place a finger on the trigger? Furthermore, upon solving the puzzle and performing the correct action, the user must learn of his or her success through a readjustment of the hand position or at best a light buzz from the controller's vibration motor or rely on visual cues. This is a far cry from the precise haptic feedback delivered to the hand by every concrete object one touches, which often gives the ability to perform tasks blindly.  The perfect controller would therefore take away the layer of abstraction posed by today's controllers between a VR user and the virtual world.

## Overview of Recent Glove-Shaped Peripheral Prototypes

Numerous companies, many new on the market, continue to put forward prototypes for glove-shaped VR peripherals. Since the release of the original CyberGlove, a glove capable of tracking the conformation of the user's hand, in 1990, CyberGlove Systems have continued until recently to evolve the product, and developed a fleet of systems based on the contemporary CyberGlove iterations.

Notable products are the CyberTouch, which delivers tactile feedback through a set of six vibro-tactile actuators, and the CyberGrasp, which delivers force feedback by manipulating an exoskeleton on the user's hand. Unfortunately, the exoskeleton construction of the CyberGrasp is criticized for being heavy at 400 grams (Burdea, 2000). Their development has become stagnant and none of the marketing material features the use of modern consumer VR systems. Conversely, newer glove-shaped peripherals have a greater focus on integration with the HTC Vive and the Oculus Rift. A promising Kickstarter project VRgluv yielded a prototype glove capable of hand tracking and force feedback through an exoskeleton. However, even after reaching its funding goal, the company is not transparent about the details of the glove's technology and there are few, if any, units in the hands of the public. Furthermore, the expected weight is similar to that of the CyberGrasp, which was formerly found problematic. The HaptX Glove prototype uses set of IMUs to track digit positions and deliver force feedback through a pneumatic exoskeleton which the manufacturer claims to be lightweight. Most impressive, however, is the HaptX Glove's further use of the pneumatic system to deliver tactile feedback through a set of 130 pneumatic actuators spread throughout the glove. A simpler offering is the VRfree prototype by Sensoryx, which relies on IMUs for hand tracking but offers no haptic feedback. While the few publicly available reviews of modern VR-oriented glove-shaped peripherals are largely favourable, the practicality, viability of construction, physical characteristics, price, and in some cases underlying technology will remain unknown until after the products are readily available to mainstream developers and the public.

## Design Overview

The prototype implements a conductive ink-based flex sensor, a conductive rubber stretch sensing cord, and a servo motor for each digit. The design decisions for each of the sensors and the servo motors are addressed in the upcoming sections of the paper. Using an elastic work glove as the

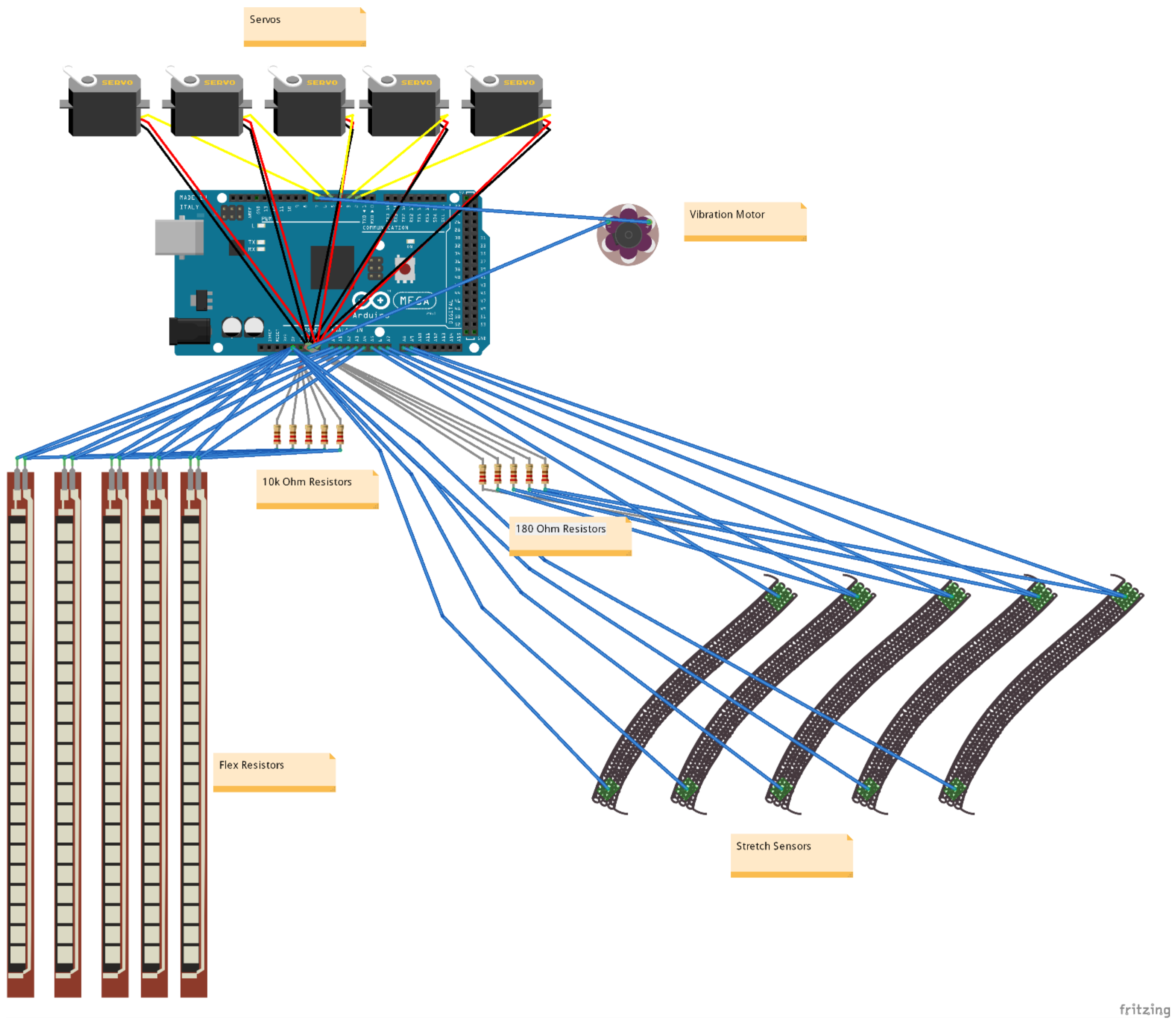


*Figure 2 Overview of prototype's circuit.*

basis for the prototype, ensures that the prototype is easy to place and adjust on the user's hand. Although a system that requires attachment of its individual components to the user's hand may achieve better accuracy, it would also be cumbersome to attach it to and detach it from the user. I attached the base of each flex sensor in front of the proximal phalanges on the outer edges of each digit. To ensure free, yet unitary movement of the flex sensors with the digits, each is surrounded with a cloth channel attached securely to the glove. The servo motors are attached and stabilized on the glove with a mounting plate designed in Google SketchUp Make 2017, and 3D printed using ABS plastic. The mounting plate consists of two pieces that are flexibly connected with wire, and sewn to the back of the glove. An earlier version of the servo mounting plate consisted of a single 100 mm wide piece, and did not adhere well to the curvature of the top of the user's hand, which resulted in the variation of distance between the fingers and the servos during operation. The size and orientation of the servo motors does not allow for the possibility to make the mount significantly narrower. Instead, the mounting plate was widened to a collective width of 120 mm to accommodate a more robust mounting architecture, and split into two pieces, yielding a degree of freedom that allows flexible adhesion to the hand. By making contact with more of points on the hand, the latest iteration of the servo mount achieves greater stability than its predecessor. An alternative solution would be mounting the servo motors elsewhere on the arm and transmitting the applied force using cables. However, this design would offer little benefit; thanks in part to the use of ABS plastic for the basis of the mount and a lack of a complex exoskeleton, the glove (excluding the Arduino board) weights 160 grams, less than half the weight of similar products which weigh as much as 400 grams, which makes weight reduction a low priority; any system used to transfer the force exerted by the relocated servo motors to the fingers would incur a loss of force and introduce undesirable elasticity in the force delivery system. Servo motors are secured using zip ties. Velcro straps that thread through loops formed by zip ties are used to secure the attachment of the servo motor mounting plate and the VR system's controller to the user's hand. The horns of the servo motors are connected to the tips of respective digits through an assembly consisting of the stretch sensors and wire. The portions of the wire connected to the servo horn doubles as the power lead for the stretch sensor, reducing the need for additional cables, while a resistor serves as a mounting point of a stretch sensor at the tip of the digit. Threading one end of the resistor through the stretch sensor and plugging each end of the resistor into a removable connector for communication with the Arduino board constructs the desired circuit, fulfils a structural role for the stretch sensor, and allows easy replacement of the fragile conductive stretch sensing rubber cord. Loops of thread and plastic channels mounted at the base of flex sensors serve to center each stretch sensor's path of movement, which is important to ensure consistency of signal between contractions of the digits. A vibration motor can be placed in the glove on the inner side of the user's wrist, where it does not interfere with the functionality of any other parts of the glove.

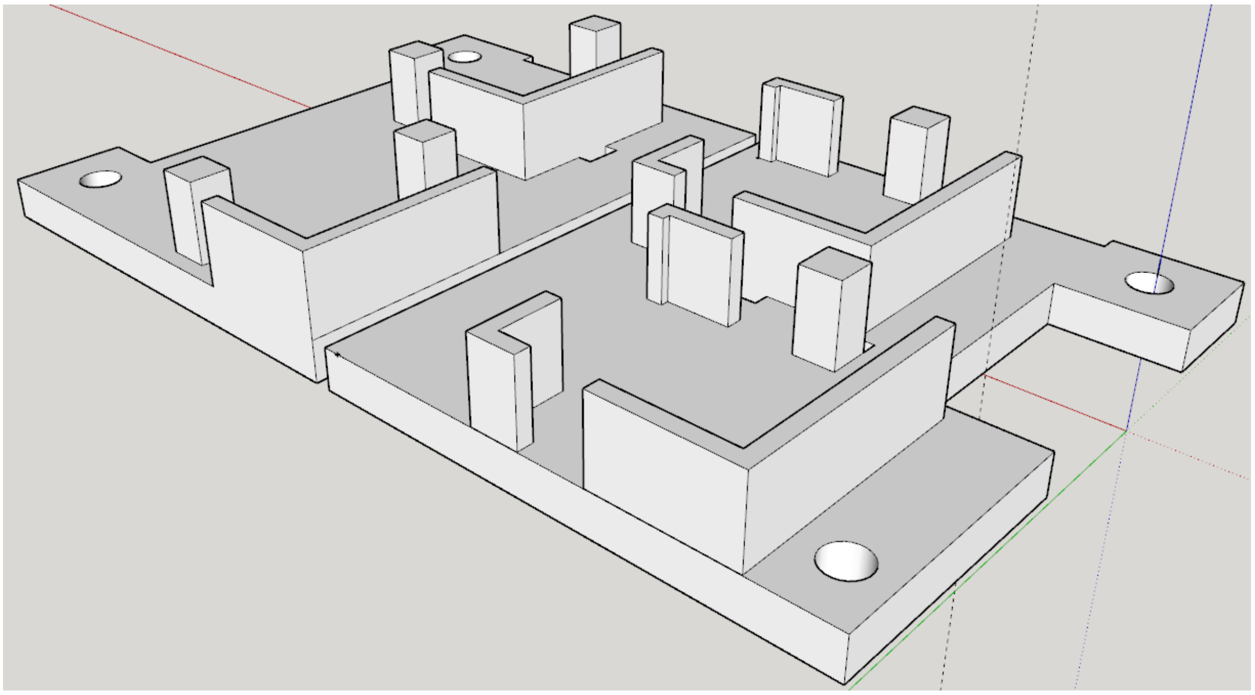


*Figure 3 Model of servo motor mounting plate.*

Data gathered from sensors and the operation of motors is managed by an Arduino Mega microcontroller board, mounted with a Velcro strap to the user's arm to avoid placing additional weight and bulk on the user's hand. For easy serviceability, the sensors are connected with double-sided female header cables to male header pins soldered to a project shield. To ensure sufficient power delivery for the servo motors, a supplemental power supply is necessary. The Arduino is connected to the host device through USB.

## Digit Tracking

Conveying the position and curvature of each finger to the host device has been a core feature of glove-shaped peripherals since the early days of the Sayre Glove. It is the foundation of a glove-shaped peripheral – without awareness of his or her hand's conformation, delivery of haptic feedback, or detection of force application would be futile for most foreseeable use cases. Therefore, it serves as a starting point for the prototype.

### Justification

Manual interactions such as picking up a pistol for combat, turning a screwdriver to repair a vehicle, or twisting the hand into a gesture to signal teammates are central to many VR experiences. While interacting with the surroundings, the user can examine the avatar's hand model, often exploring the steps that have been taken to make it more believable. Additionally, each interaction calls for unique motions and hand positions. Because of the differences in their size and shape, holding a revolver would feel vastly different from holding a smartphone. The level of abstraction of the environment posed by common controllers puts a choice before the developers between making each interaction as unique in VR as it is in the real world or using an agreed-upon control scheme. While the first approach can make for a more engaging experience, it necessitates the user learning a separate sequence of buttons to press and triggers to squeeze for every interaction. Conversely, applying a generalized control strategy to all interactions reduces the user's learning curve at the cost of removing him or her from the diverse geometries of the virtual objects he or she touches. An accurate representation of the hands removes the burden of this choice as it gives the developers the opportunity to take advantage of realistic hand conformations for interactions without requiring a steep learning curve for the user as the motions called for by the experience are learned in day-to-day life.

Realistic representation of the behaviour of one's digits in the virtual world would therefore greatly increase the believability and intuitiveness of an experience.

## My Implementation

To asses digit flexion, my glove uses a set of five conductive ink based flex sensors, one for each finger. To reduce the physical stress near the connective leads and ease connection to Arduino with female-to-female connector cables, the sensors are linked to male header pins supported by a piece of flexible breadboard. To unite the movement of the flex sensor with the finger, the sensor is loosely placed within a cloth channel sown to the glove's fingers where it can freely move along the digit while being forced to follow its flexion.

Each assembly is mounted on the outside of the glove at the base of the digit's proximal phalange. The flex sensors are positioned over the proximal interphalangeal joints of the fingers, giving the glove 1 DOF tracking on each finger. The proximal interphalangeal joint was chosen for its nearly 135° range of movement and involvement in most finger movement. While the metacarpophalangeal joint offers a similar range of motion, common hand positions have a lesser effect on its angle. The distal interphalangeal joint has around 30° range of movement and its position is closely linked to that of the proximal interphalangeal joint. The interphalangeal joint is chosen on the thumb for a similar reason.

Once the sensors are connected to Arduino's analog input pins, the data can be read and stored in integer variables. The beginning of the Arduino's sketch code globally declares each digit an integer variable for storing the value read from the flex sensor, and a constant integer to hold the value of the analog pin on the Arduino associated with each flex sensor. For example, such declarations for the index finger flex sensor are as follows:

```
int flxInd;

const unsigned int FLEX_2_PIN = 2;
```

Within `void Loop()`, the Arduino calls the function `readFlex()`, which reads the potential of the analog pins to which the flex sensors are connected and stores the resulting value in the appropriate variable.

```
void readFlex(){
        flxThu = analogRead(FLEX_0_PIN);
        flxInd = analogRead(FLEX_1_PIN);
      …
}
```

After this, the collected data in the integer variables is collected into a string packet which is sent over the serial connection to the host device. The digit positions of the hand model can be calculated from the flex sensor data in the 3D engine.

## Overview of Other Implementations

Digit tracking has been a fundamental goal of glove-shaped peripherals since the early days. Over time, manufacturers have taken a myriad of different approaches to fulfil this goal. VPL's DataGlove uses optical flex sensors to capture the angles of different joints. Thanks to the simplicity of implementation, different types of flex sensors continue to be abundant in similar peripherals, used similarly in the Nintendo Power Glove, and the CyberGlove family of devices. Prototypes from Exos use hall effect sensors, although the company's design documentation proposes the use of potentiometers to track the conformation of an exoskeleton that encompasses the user's fingers as another option. Most recently, prototypes from HaptX and Sensoryx use IMUs to track the behaviour of digits.

## Discussion

### Achieving Comprehensive Tracking

A fundamental limitation of my design for finger tracking is the lack of tracking of the metacarpophalangeal joint positions. While it is true that the proximal interphalangeal joints are heavily involved in most activities undertaken by hands, the user can control the orientation of the metacarpophalangeal joint independently from the rest of the hand with respect to two degrees of freedom, making it necessary for building an accurate hand conformation model. Tracking a metacarpophalangeal joint poses a level of difficulty above that of a proximal interphalangeal joint because besides flexion it is also involved in lateral extension of the digits. The challenge extends to the trapezometacarpal joint found at the base of the thumb which similarly offers two degrees of freedom.

One approach to achieving a more comprehensive hand tracking model is using more flex sensors to capture all the intricacies of the hand's conformation. Because each flex sensor can only capture a single degree of freedom, accounting for each independently movable element of the hand can require upwards of 20 sensors. For example, iterations of VPL Research's DataGlove use data from 16 flex sensors, the latest CyberGlove is offered with up to 22 flex sensors. Alternatively, using finger-mounted IMUs greatly simplifies the construction of the glove. Sensoryx VRFree glove uses 11 IMUs, one on each of the interphalangeal joints, and one on each joint of the thumb. It would, however, be sufficient to use 8 6 degree of freedom IMUs, one placed on each of the fingers' middle phalanges, one on each of the thumb's phalanges, as both can be operated independently, one on the palm, and one on the user's wrist. The positional and rotational data from each IMU can then be used to infer the states of any points that are not tracked directly but are coupled to parts of the hand whose location and rotation are known.

## Sensor Choice and Signal Quality

Even when the flex sensors are left undisturbed, the analog signal read by the Arduino is prone to fluctuation. Nisar et al. propose that the signal can be stabilized by adding a small-value, non-polar capacitor in parallel with the conductive ink based flex sensors to reduce the ripples from the power source (2014). Furthermore, they emphasize the importance of reliable attachment of the sensor to the glove and thermal insulation (Nisar et al., 2014). A Kalman filtering algorithm can be used to reduce signal noise in software. Better results may also be found by using other types of sensors. The VPL

Research DataGlove makes use of fiber optic flex sensors. Meanwhile, Exos, who engineered a series of glove-shaped peripherals with exoskeletons, propose a series of approaches that include monitoring positions of joints of the glove's exoskeleton with potentiometers, placing hall effect sensors on the exoskeleton, and inferring the conformation of the digit from the position of the motor used for haptic feedback. When IMUs are used for finger tracking, repeatability, stability, and drift are other factors that are important to consider ("IMU Errors and Their Effects"). Besides structural considerations, choosing the best sensor finger tracking necessitates further comparison of signal consistency, noise, and fidelity.

## Haptic Feedback

While hand tracking is a valuable first step, and is often sufficient for entertainment applications, haptic feedback widens the applicability of a glove-shaped peripheral. VR applications across many industries stand to gain from an accurate representation of virtual objects. Furthermore, user force detection, discussed in the next section of the paper, is reliant on an opposing force being exerted by a haptic feedback system. As such, this is the next piece of the glove's design that must be implemented.

### Justification

In accordance with Newton's third law, which states that for every action there is an equal and opposite reaction, everything we touch responds with a similar force. Thanks to this property of the physical world, people rely heavily on the haptic senses. Without looking, one can know whether he or she has successfully picked up an apple, pressed a key on a remote, or crushed an empty can. Except for the unchanging feel of its buttons and triggers, and rudimentary vibration feedback, today's common VR controllers do little to convey the rich haptic experience of physical interactions. Sutherland proposes a kinesthetic display consisting of levers that variably resist forces to give the user force feedback (1965). Project GROPE demonstrated the viability of Sutherland's concept by comparing subjects' performance while using interfaces with and without force feedback as they participated in a series of abstract educational tasks such as finding optimal docking of a molecule in a protein's reactive site (Brooks et al., 1990). The multidecade project revealed that force feedback gives the participant an edge (Brooks et al., 1990). Because haptic feedback is a major component of people's perception of the world, immersiveness and educational potential of VR experiences would stand to gain from a haptic feedback system that dynamically and believably conveys forces, forms, and tactile feel of virtual world.

## My Implementation

### Form Depiction

To convey the form of a virtual object, I use a set of servo motors mounted to the back of the hand, one for each digit. The motors' horns are coupled to the user's fingertips through electrical cable and a piece of conductive rubber stretch sensing cord. To ensure that the connective assembly travels along the outer edge of each digit consistently, it is threaded through plastic tube that is mounted at the base of the flex sensor and extends past the proximal interphalangeal joint and stabilized near the tip of the digit with a loop of thread. To depict the approximate shape of a virtual object, the servo motors retract the connective cables, individually limiting the inward flexion of each digit.

I chose servo motors to depict haptic feedback because they are easy to position precisely, can be lightweight, and require minimal external equipment to operate. Regular constant rotation motors are not easy to position precisely. Stepper motors are bulkier and offer no advantage over a servo motor in the given application. A recurring approach in the glove-shaped peripheral industry is the use of pneumatics. While the ability to easily apply force over a wide area and move the actuator away from the hand makes this option tempting, because of the thermal, barometric, and volumetric instability of air, a pneumatic system can be cumbersome to design in a way that delivers a consistent haptic response. Furthermore, a pneumatic system typically requires an external unit and a complex control structure to manage inflation of individual air pockets, which can make such a system unwieldy.

The position of each servo motor can be dictated by the Arduino's pulse width modulation (PWM) capable digital pins. Arduino IDE's built-in libraries abstract the PWM interface used to communicate with a servo motor to calling a function with the parameter of the desired horn angle in degrees. First, it is necessary to include the appropriate library with the statement:

```
#include <Servo.h>
```

Then the Arduino's sketch code globally declares the pin associated with each servo, the `Servo` object, and an integer variable to hold the desired servo angle for each digit. Such a declaration for the middle finger is as follows:

```
const unsigned int SERVO_2_PIN = 4;
Servo serMid;
int sersMid;
```

Within the Arduino's `setup()` function, each servo object is linked to a unique PWM pin. For instance, the middle finger servo motor is initiated by calling:

```
serMid.attach(SERVO_2_PIN);
```

Upon initiation, the servo motor becomes immobilized and maintains the position dictated by the PWM signal. Because of this, the servos' built-in potentiometers cannot be used to infer finger position.

In each iteration of the Arduino's `loop()` function, each servo's desired position is read from the data packet received from the host device and stored it the respective integer variable, a topic covered in the section of this paper on serial communication. Then, each servo is directed to the desired position. For example, the middle finger servo is directed to the angle in the variable `sersMid` by calling:

```
serMid.write(sersMid);
```

This function alters the PWM signal delivered to the servo's data connection to signal the desired angle.

## Vibration Feedback

The vibration motors found in common VR controllers give the user a useful degree of haptic feedback, indicating that he or she has successfully fired a pistol or picked up an object. To continue to take advantage of this mechanic, I implemented a single vibration motor in the glove-shaped peripheral. The vibration motor can be turned on and off by applying a high or low voltage to the digital pin it is connected to on the Arduino. First, the pin responsible for operating the vibration motor and a Boolean variable that controls the state of the vibration motor are globally declared:

```
const unsigned int VIBRATE_PIN = 7;

bool vibrate;
```

Then, within the Arduino's `start()` function, the digital pin's mode must be set:

```
pinMode(VIBRATE_PIN, OUTPUT);
```

When the Boolean variable responsible for manipulating vibration is enabled through the serial port, a high signal is applied to the pin that powers the vibration motor. Else, a low signal is applied.

```
if(vibrate == true)

       digitalWrite(VIBRATE_PIN, HIGH);

else

       digitalWrite(VIBRATE_PIN, LOW);
```

## Overview of Other Implementations

Over time, two primary approaches to providing the user force feedback emerged – exoskeletons and pneumatics. Exos' prototypes, CyberGlove's CyberGrasp, and the more modern VRgluv exert force to the user's hand through an exoskeleton system which provides robust and direct control over respective joints. Other systems, notably the Teletact II and a novel rehabilitation system designed by Connely et al. (2010) inflate pneumatic pockets to achieve a similar effect. HaptX draws from both design paradigms in a design that involves a minimal, pneumatically actuated exoskeleton.

Less common is the implementation of tactile feedback. In similar designs, CyberGlove's CyberTouch and the TactGlove use vibro-tactile actuators and piezo sounders, respectively, to deliver rudimentary tactile feedback on the user's fingers. Although little is yet known about HapteX's tactile feedback system which takes advantage of pneumatic actuators, the prototype appears to provide the most comprehensive tactile feedback system available.

## Discussion

### Limits of Shape Depiction

Upon first inspection, my approach to portraying the shape of a virtual object is dwarfed by past endeavours. My system only offers a single degree of freedom and only limits inward flexion for each digit. Meanwhile Exos' SAFIRE single-finger prototype can manipulate the operator's finger with respect to two degrees of freedom, and CyberGlove's CyberGrasp offers three degrees of freedom for each digit (Eberman, 1993). However, each of these systems relies on an exoskeleton to deliver the actuators' forces, which adds weight, bulk, and potential points of failure to the system. Teletact II takes a different approach to reflecting shapes and forces by placing 30 pneumatic pockets around the hand (Stone, 2000). This method, however, requires a fast, complex pneumatic system to inflate and deflate the air pockets at a rate that would make the experience believable. A pneumatic system adds complication, expense, and bulk to the product. My system, therefore, is uniquely simple.

### Flex in Connective Assembly

The conductive rubber stretch sensing cord within the connective assemblies of the prototype Easily stretches with little force, which detracts from the rigidity of the depicted shape. Lack of a rigid attachment of the plastic tubes through which the connective assemblies are routed further contributes to the undesired effect. The first factor can be mitigated by using thicker or more rigid stretch sensing cord for force detection. If such a conductive material cannot be obtained, the stretchable part of the connective assembly a piece of more rigid non-conductive material can also be attached alongside. Both factors could further be minimized by routing the connective assemblies through a channel formed with solid plastic rings and thimbles for each finger.

### Vibration and Tactile Feedback

The haptic experience would further be enhanced with realistic engagement of tactile senses. Haptx's glove delivers an impressive and comprehensive depiction of tactile feedback using 130 pneumatic chambers distributed throughout the glove which can be inflated to stimulate the sense of touch. An easier-to-implement method, carried out by CyberGlove's CyberTouch and the TactGlove, is placing vibration motors or piezo sounders in the palm and finger assemblies to increase the resolution possible with vibration feedback.

## Force Detection

To maximize the benefits of a glove-shaped VR system and the believability of the experience, a dedicated force detection system is highly beneficial. Reliant on hand tracking and a force feedback system, it must be implemented as a final step.

### Justification

Forces lie at the core of the haptic experience. We must exert forces with our hands to pick up, crush, and press the objects around us; merely placing our hands around objects is usually insufficient to achieve these actions. A haptic display, as described in the previous section, conveys the forces sustained by the avatar's hand without directly capturing the forces the user exerts with his or her hand in response. This limits the representation of objects that deform under pressure (Budea, 2000). If one applies sufficient force to an empty tin can, it collapses. Likewise, a button that is being pressed first meets the user with a degree of resistance before the switch caves to the pressure. To accurately represent the composition of a can or the tactile feedback of a button in a virtual space, the user's interface must reflect the forces applied by each digit.

## My Implementation

To asses the forces applied by each digit, I use a 4 cm piece of conductive rubber stretch sensing cord within the junction between servo horns and the tip of the user's finger. Each end of the conductive rubber cords is supported by the cables that, besides having a structural role, link it to the Arduino. I chose to use stretch sensors rather than pressure sensors because ensuring consistent and detectable force detection on a pressure sensor would require additional structure at the fingertips. To add support and achieve consistent behaviour, the conductive rubber cords are stabilized and centered near the tip of each digit with a loop of thread. As the user resists the hand position imposed by the prototype's form depiction system, he or she stretches the conductive rubber cord, altering its length, and consequently the electrical potential read by the analog pins of the Arduino.

To take advantage of the stretch sensors, Arduino's sketch globally declares a constant that determines the analog pin where the signal is read, and an integer variable that stores the signal read from the stretch sensor. An example declaration for the ring finger's stretch sensors is:

```
const unsigned int STR_3_PIN = 8;

int strRin;
```

To read the data from each stretch sensor, the Arduino calls the following function within its update loop:

```
void readStretch() {

        strThu = analogRead(STR_0_PIN);

        strInd = analogRead(STR_1_PIN);

      …

      }
```

Lastly, the data from the stretch sensors is added to the information packet that is passed to the host device over serial connection.

### Overview of Other Implementations

The VRgluv uses pressure sensors to reflect grip strength to the host device. None of the other prototypes I reviewed used had sensors dedicated to force detection. However, it must be noted that some early telemanipulation systems discern force based on the difference between the positions of the master manipulators and the slave actuators. It is possible for existing systems with force feedback to take a similar approach, calculating the force applied by the user based on the error between the existing hand conformation and the one imposed by the force feedback system.

### Discussion

The signal yielded by the prototype's stretch sensors is obstructed by the noise fluctuations. A more consistent signal may be achieved by using a small-value, non-polar capacitor in parallel with the sensor, an approach suggested by Nisar et al. for improving the signal from flex sensors (2014). Kalman filtering may also help mitigate this issue. Variance of the sensor's movement through its support structures further detract from signal consistency. The later can be remedied with the use of a thimble to attach the sensor to the tip of the finger and a channel for the cable to pass though. Using cables made of different, and as per the demands dictated by the needs of the force depiction system, more rigid material is also an option worth exploring.

An alternative approach is substituting the existing connective assembly with a more rigid, strictly structural coupling between the tip of the finger and the servo, implementing a structural thimble at the tip of the finger, and using a pressure sensor to asses the force that arises between the fingertip and the thimble. This solution has a dual benefit of enhancing the structural rigidity of the force depiction system, and delivering a reliable assessment of the force applied by the user's hand.

## Serial Connection to Host

Now that all of the sensors are ready to be read, and all of the motors are ready to receive signals, a system must be implemented to take full advantage of the glove-shaped peripheral. In this implementation, the prototype is used to interact with VR environment designed in Unity3D. To facilitate two-way communication, the Arduino and the relevant script in Unity3D communicate through a common USB serial port on the host computer. The details of this communication are described below.

## The Arduino Serial Interface

### Starting a Serial Link

The default Arduino libraries supply the methods necessary handle serial communication. Before starting a serial link, the prototype's Arduino sketch declares the baud rate, and strings to hold the input and output values:

```
const unsigned int BAUD_RATE = 9600;

String output;

String input;
```

The setup function then changes the timeout to 25 ms, an operation necessary as the default timeout value for a serial connection is 1000 ms, which noticeably delays the Arduino's operation when a host does not have a data packet ready, often giving rise to timeout exception errors. Then, the setup function begins the serial connection using the predetermined baud rate:

```
Serial.setTimeout(25);

Serial.begin(BAUD_RATE);
```

### Data Packaging and Exchange

The prototype's serial connection packages the data to be exchanged between the host device and the Arduino in strings. The following function, called during the Arduino's update loop, concatenates the signal from each flex sensor and stretch sensor to a single string, separated by commas:

```
void generateGloveData(){
```

```
            output = "";


            output = output + flxThu;

            output = output + ',';

              …

            output = output + strRin;

            output = output + ',';

            output = output + strPin;

        }
```

The resulting string is then sent to the host by calling:

```
        Serial.println(output);
```

The string received from the host that dictates the positions of the servo and vibration motors is delivered in a similar format. To read and unpackage it, the Arduino uses the following function called in its loop function:

```
void readServoData(){

    if (Serial.available()){

  input = Serial.readString();


  int commaPosition;

  String readNumberStorage;

  int count = 0;


  do

        {
            commaPosition = input.indexOf(',');


                readNumberStorage = input.substring(0, commaPosition);

                input = input.substring(commaPosition + 1, input.length());


                switch(count)

                {

                    case 0:

                        sersThu = atoi(readNumberStorage.c_str());

                        break;

                    case 1:

                        sersInd = atoi(readNumberStorage.c_str());

                        break;

                    …

                    case 5:

                        vibrate = atoi(readNumberStorage.c_str());

                        break;

                }


            count++;

        } while (commaPosition >= 0);

  }

}
```

## Unity3D Serial communication

### Establishing a Connection

Before a serial connection can be established in Unity3D, it is necessary to change the “Api Compatibility Level*” option in PlayerSettings to “.NET 4.x”. After this, the script responsible for handling the serial connection must include the library:

```
using System.IO.Ports;
```

Lastly, the script must declare a `SerialPort` IO stream:

```
const string PORT = "COM3";

const int BAUD_RATE = 9600;

SerialPort stream = new SerialPort(PORT, BAUD_RATE);
```

While the port name is retained between Arduino sessions with the same host, it can be changed between hosts; when this happens, the correct port name can be found in the Device Manager after connecting the Arduino.

In the `Start()` function, the script starts the serial connection and opens the IO stream:

```
stream.ReadTimeout = 10 000;

 stream.Open();
```

A timeout time of 10 000 ms helps avoid timeout exception errors and is permissible because, as explained in the next subsection, the serial communication occurs in a separate thread.

In the `Update()` loop, a function that performs data exchange with he Arduino is assigned to the thread:

```
serialComThread = new Thread(talkToAdruino);
```

### Multithreading for Serial Communication

For most operations, Unity3D uses a single thread of operations. Serial communication with an Arduino cannot achieve a rate that reaches the framerates of 60-90 frames per second desirable for VR experiences. As a result, exchanging data with the Arduino in Unity3D’s main thread can cause unusable framerates below 1 frame per second. One approach to solving this issue is using a separate thread to handle exchange of data over the serial port in parallel with the main thread. To achieve this, the script responsible for exchanging data with the Arduino includes the library:

```
using System.Threading;
```

Then, the script globally declares the global variable of the Thread type and a Boolean variable to serve as a flag that an instance of the thread is running.

```
private Thread serialComThread;
private bool threadRunning;
```

A Boolean flag is set to true in the beginning of the method performed by the thread and set to false once the function’s operation is complete. It is used to ensure that new instances of the thread are started before an existing instance is complete, a scenario which can overload the processing capacity of the host hardware by opening an excessive number of parallel threads, leading to a forced closure of the Unity3D editor:

```
 if(!threadRunning)
      serialComThread.Start();
```

Additionally, the script encapsulates any operations that access the same variables as the thread in

```
if(!threadRunning) {…}
```

This practice prevents race conditions between parallel threads. Race conditions can, for example, be problematic if the thread tasked with exchanging information with the Arduino sends a string data packet before it is concatenated to include all information, thus sending an incomplete set of data to the Arduino.

The method invoked in the serial communication thread is as follows:

```
private void talkToAdruino()
```

```
     {
          threadRunning = true;
          serialInput = stream.ReadLine();
          stream.WriteLine(serialOutput);
          threadRunning = false;
    }
```

While inspired for use in the project to mitigate delays in serial communication, threading is commonly used to perform time-consuming calculations with a lesser impact on the program's performance. For instance, the prototype's script can further be optimized by using a secondary thread to process data packages, and map values.

## Data Packaging

Consistent with the format in the Arduino's sketch, the script responsible for glove behaviour in Unity3D exchanges data packets in the form of strings. While differences between the methods for handling strings found in Unity3D and Arduino IDE necessitate different syntax, similar techniques are used to disassemble and concatenate information packets. The string that stores serial input, `serialInput`, is read in the following manner:

```
int commaPosition;
do
{
        commaPosition = serialInput.IndexOf(',');
        string readNumberStorage = serialInput.Substring(0, commaPosition);
        serialInput = serialInput.Substring(commaPosition + 1);
        switch (count)
                {
                        case 0:
```

```
                        flxThu = int.Parse(readNumberStorage);
                        break;
                ...
                    case 9:
                        strPin = int.Parse(readNumberStorage);
                        break;
                }
        count++;
} while (commaPosition >= 0);
```

The string that stores serial output, `serialOutput`, is concatenated using:

```
        serialOutput = "";
        serialOutput = serialOutput + serThu;
        serialOutput = serialOutput + ',';
        …
        if (vibrateOn)
                serialOutput = serialOutput + 1;
        else
                serialOutput = serialOutput + 0;
```

## Hand Model Manipulation

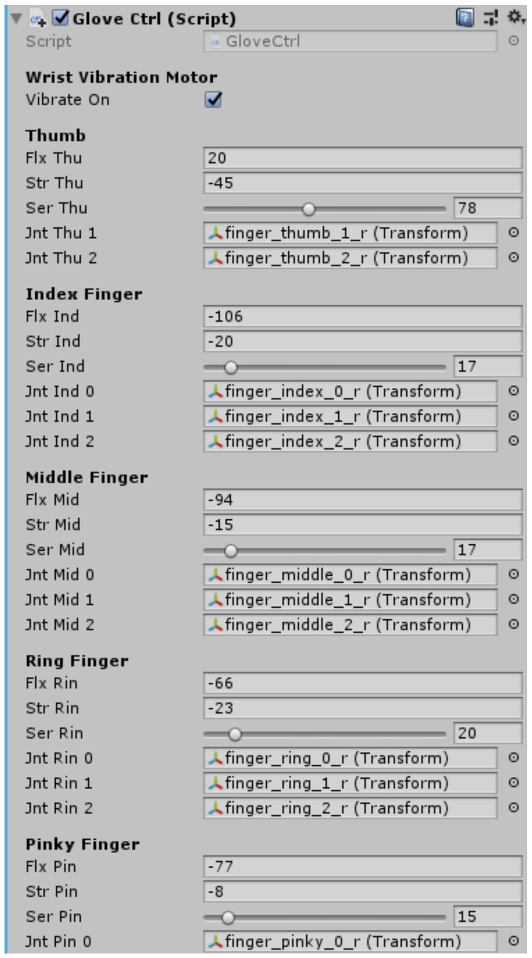
The script responsible for handling glove behaviour applies the data to a hand model supplied by the SteamVR asset. After disabling the animator and SteamVR behavioural scripts, it is possible to take advantage of the hand model's skinned mesh renderer and representation of pivotal points of the hand as empty child game objects to shape the hand model's conformation. To facilitate spatial tracking, the hand model is imported as a child to the right hand anchor of the VR camera rig system in use. The script, placed under the hand model's parent object, takes reference to the `Transform` parameters of the pivotal game objects representative of digit joints, and locally rotates them around the Z axis based on data read from flex sensors. For instance, the thumb's rotation is manipulated by rotating the objects that represent the game objects that represent its metacarpophalangeal and interphalangeal joints:

```
flxThu = Map(flxThu, 250, 90, 0, -90);
jntThu1.localRotation = Quaternion.Euler(0, 0, flxThu);
jntThu2.localRotation = Quaternion.Euler(0, 0, flxThu);
```

Before a rotation position can be set, the flex sensor data is mapped using a function adapted from the Arduino IDE library in a manner that a range of 250 to 90 is rescaled to a range of 0 to -90, realistically representing the curvature of a digit in each joint. The values used can be altered to calibrate the relative flexion of each join on the hand model. The script targets the quaternion `localRotation` to achieve this task because it allows setting the rotation of a game object relative to its parent. Conversely, the quaternion `rotation` sets the rotation of the game object relative to the world space, while the method `Rotate(float xAngle, float yAngle, float zAngle, Space relativeTo = Space.Self)` applies a rotation to the object every time it is invoked rather than setting it to the desired value; while both alternative options can be used reach the desired effect, they would unnecessarily additional code to implement. The `Euler(float x, float y, float z)` method in the `Quaternion` class trivializes the setting of rotation to developers who prefer to use Euler angles rather than quaternions by converting rotation parameters to the later format.

## Application within VR Experience

Reading the states of sensors and setting the states of motors through the script that handles glove behaviour gives prospective developers numerous possibilities for interactions in VR. To demonstrate the application of the prototype I implemented a rudimentary object grabbing system. A contact of a collider on the hand model with a virtual object triggers the retraction of user's digits using servo motors to give the haptic sensation of wrapping his or her hand around an object. The system can be altered to set the servos to different positions depending on the form of the object. A more elaborate system can also be developed to trigger the retractive action of servomotors when colliders on inner



**Glove Ctrl (Script)**

| | |
|---|---|
| Script | GloveCtrl |
| **Wrist Vibration Motor** | |
| Vibrate On | ✓ |
| **Thumb** | |
| Flx Thu | 20 |
| Str Thu | -45 |
| Ser Thu | 78 |
| Jnt Thu 1 | finger_thumb_1_r (Transform) |
| Jnt Thu 2 | finger_thumb_2_r (Transform) |
| **Index Finger** | |
| Flx Ind | -106 |
| Str Ind | -20 |
| Ser Ind | 17 |
| Jnt Ind 0 | finger_index_0_r (Transform) |
| Jnt Ind 1 | finger_index_1_r (Transform) |
| Jnt Ind 2 | finger_index_2_r (Transform) |
| **Middle Finger** | |
| Flx Mid | -94 |
| Str Mid | -15 |
| Ser Mid | 17 |
| Jnt Mid 0 | finger_middle_0_r (Transform) |
| Jnt Mid 1 | finger_middle_1_r (Transform) |
| Jnt Mid 2 | finger_middle_2_r (Transform) |
| **Ring Finger** | |
| Flx Rin | -66 |
| Str Rin | -23 |
| Ser Rin | 20 |
| Jnt Rin 0 | finger_ring_0_r (Transform) |
| Jnt Rin 1 | finger_ring_1_r (Transform) |
| Jnt Rin 2 | finger_ring_2_r (Transform) |
| **Pinky Finger** | |
| Flx Pin | -77 |
| Str Pin | -8 |
| Ser Pin | 15 |
| Jnt Pin 0 | finger_pinky_0_r (Transform) |

*Figure 4 Graphical interface for script responsible for the prototype's behaviour.*

surfaces of individual digits come in contact with virtual surfaces. Once the conformation of the user's hand is constrained to reflect the grabbable object, the stretch sensor signals are monitored to asses whether the user is applying sufficient force to grab the object. Once the threshold of force is reached, the object is picked up by becoming the child of the hand model object and the user is informed of his or her success with the engagement of the vibration motor. A drop of the stretch sensor signal below the threshold needed to pick up the object prompts the release of the object and a disengagement of the vibration motor. When the stretch sensor signal is not sufficiently stable to accurately reflect the force applied by the user, as was the case in the testing of the prototype, finger curvature read from flex sensors can be read instead. Once the object's collider ceases to interact with the hand model's collider, the servo motors are adjusted to the position least restrictive on hand operation.

## Applications of Technology

The technology found in the prototype has uses far beyond VR entertainment. While tracking and representing the behaviour of digits serves as a foundation for any VR implementation of a glove-shaped peripheral, it also has applications on its own; with he help of gesture recognition, hand tracking can facilitate unique communication systems and new opportunities to learn new skills such as sign language (Weissmann & Salomon, 1999). The addition of haptic feedback widely expands possibilities in diverse industries. Accurate representation of virtual objects' shapes can improve user's performance in learning about abstract concepts such as force fields and molecule docking, computer assisted design, robot-assisted surgery, and telepresence scenarios (Brooks et al., 1990; Burdea, 1993; Okamura, 2009). Force detection, although seldom implemented with discrete sensors, can enhance existing implementations of force-feedback systems. Connelly et al. (2010) designed a rehabilitative glove that is capable of hand tracking and uses a pneumatic system to assist stroke patients in finger extension, an operation that the group often finds challenging. Allin, Matsuoka, & Klatzky (2002) use a desktop force measurement system to evaluate self-imposed limits on force production, which are hypothesized different among stroke victims. By enhancing the rehabilitative glove with a precise force detection system, the assistive force exerted by the glove can be gauged more accurately, and the force applied by the patients can serve as a valuable parameter for rehabilitation progress. Research reviewed Okamura (2009) demonstrates that force detection and representation can improve the performance and safety of robot-assisted surgery. While the prototype I describe is far from the level of refinement that would make it beneficial in most of these scenarios as is, continued development and intertwining with existing designs can yield a series of products with potential in the fields of education, medicine, engineering, and design.

## Conclusion

Manual interaction is a core part of modern VR experiences. A peripheral with accurate digit tracking, haptic feedback, and force detection would be an important step to maximizing the believability and impressiveness of a VR system. In this project, a working prototype of such a peripheral was built and tested. Some problems remain including but not limited to a lack of tracking of the position of metacarpophalangeal joints, a lack of rigidity in the haptic feedback system, and an unusably low signal to noise ration of stretch sensor. Nonetheless, the prototype acts as a proof-of-concept for a glove-shaped peripheral for VR.

## References

Allin, S., Matsuoka, Y., & Klatzky, R. (2002). Measuring just noticeable differences for haptic force feedback: implications for rehabilitation. In Proceedings 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. HAPTICS 2002 (pp. 299-302). IEEE.

Brooks Jr, F. P., Ouh-Young, M., Batter, J. J., & Jerome Kilpatrick, P. (1990, September). Project GROPEHaptic displays for scientific visualization. In *ACM SIGGraph computer graphics* (Vol. 24, No. 4, pp. 177-185). ACM.

G. Burdea, Virtual Reality Systems and Applications," Electro' 93 International Conference, Short Course, Edison NJ, April 1993.

Burdea, G. C. (2000). Haptics issues in virtual environments. In Proceedings Computer Graphics International 2000 (pp. 295-302). IEEE.

Connelly, L., Jia, Y., Toro, M. L., Stoykov, M. E., Kenyon, R. V., & Kamper, D. G. (2010). A pneumatic glove and immersive virtual reality environment for hand rehabilitative training after stroke. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *18*(5), 551-559.

Nisar, O. (2014, May*). Performance Optimization of a Flex Sensor Based Glove for Hand Gestures Recognition and Translation, International Journal of Engineering Research & Technology*, Vol. 3, Issue 5. Faisal Town, Lahore, Pakistan.

Novatel. (n.d.). *IMU Errors and Their Effects*. Retrieved April 11, 2019, from https://www.novatel.com/assets/Documents/Bulletins/APN064.pdf.

Okamura, A. M. (2009). Haptic feedback in robot-assisted minimally invasive surgery. Current opinion in urology, 19(1), 102.

Pollack, A. (1989, April 10). For Artificial Reality, Wear A Computer. Retrieved from https://www.nytimes.com/1989/04/10/business/for-artificial-reality-wear-a-computer.html

Rosson, L. (2014, April 15). The Virtual Interface Environment Workstation (VIEW), 1990. Retrieved from https://www.nasa.gov/ames/spinoff/new_continent_of_ideas/

Stone, R. J. (2000, August). Haptic feedback: A brief history from telepresence to virtual reality. In *International Workshop on Haptic Human-Computer Interaction* (pp. 1-16). Springer, Berlin, Heidelberg.

Sturman, D. J., & Zeltzer, D. (1994). A survey of glove-based input. *IEEE Computer graphics and Applications*, *14*(1), 30-39.

Sutherland, I. E. (1965). The ultimate display. *Multimedia: From Wagner to virtual reality*, 506-508.

Weissmann, J., & Salomon, R. (1999). Gesture recognition for virtual reality applications using data gloves and neural networks. In IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339) (Vol. 3, pp. 2043-2046). IEEE.