Virtual Reality Force Feedback and Safety Device

Final Documentation

Good Good Boys:

Stephen Siemonsma, Kevin Mattes, Sam Hisel, and George Drivas

May 6, 2019

1) Introduction

Virtual reality (VR) is a set of technologies that are rapidly evolving into a prominent and mainstream feature of the video game and entertainment industries. The Facebook-backed company Oculus is one of the most prominent companies innovating in this field. Their first consumer headset, the Oculus Rift, with its fully tracked headset and controllers allows for near-limitless creative uses for both consumers and developers. However, the Oculus Rift and all the other systems on the market suffer from glaring safety omissions that make their use hazardous for even the most experienced of users. A VR head-mounted display (HMD) fully obscures the user's vision and immerses them in a virtual world, blinding them to the outside world and often causing them to completely forget about their physical surrounding. Although "virtual fence" software implementations exist for these systems to alert the user when they are in danger of exiting their safe VR play area, these systems are often ineffective with smaller play spaces, rapid in-game movements, and less experienced users. Consequently, a user can accidentally strike walls, objects, and even other people while immersed in their simulated world. This can easily result in injuries and physical damage to their surroundings.

To alleviate some of the safety deficits of current virtual reality products, we developed a hardware prototype and software solution that is able to physically prevent the user from extending their arms outside of their established play area. This includes a wearable vest system and rope-attached wrist guards that are able to engage variable levels of resistance to outward arm movement. The full capabilities of this system are showcased in a virtual reality demonstration program featuring a variety of virtual objects and an enforced play space boundary. Although physically restraining outward arm movement for safety purposes was the primary focus of this project, in-game collisions with virtual objects are also augmented by the force feedback system. The demonstration program strongly leverages the Oculus Rift's tracking system in order to make sophisticated and predictive decisions about when and how to signal the vest unit to engage resistance. The demonstration program runs on the same PC that powers the Oculus Rift virtual reality headset. Communication between the PC and the Raspberry Pi-controlled vest unit occur over the local Wi-Fi network using UDP packets.

2) Project Outcome

We are overall very satisfied with the outcome of this project. We met or exceeded almost all of the functionality objectives, requirements, and constraints from our original project proposal. We even added extra features such as the analog-to-digital converter that allowed us to monitor the battery level. One goal that we were not able to meet was the 100 lbs. maximum force capability. In reality, we know our braking system is capable of this kind of force, but we did not have time to adjust the final hardware protype once we were very close to the Modern Marvels event. Additionally, it is possible this kind of force would jeopardize the integrity of weaker aspects of the system such as the fabric loops that attach the ropes to the wrist guards, something that we could not tolerate failing last-minute. However, even with only ~40 lbs. of force, the full functionality of the device was surprisingly good and fun to use. It was able to keep the user's hands within the play space boundaries, even during faster hand movements. Additionally, the 3 main demonstration stations worked well for demonstrating the various capabilities of the force feedback system, with the deformable ball station exceeding even our own expectations. Lastly, we were able to integrate a level of polish with our menu system and heads-up notifications that made us very proud of the end product.

Over the course of the project, the team member roles have been largely consistent with what we detailed in the original proposal. Stephen was involved in all aspects of the software implementation and testing. This work was primarily focused on the Unity demonstration program and all of the underlying logic regarding the behavior of the force feedback system. Kevin worked primarily on the networking aspects of the project. Working between the device Raspberry Pi and the Unity game demonstration, he was able to successfully implement an asynchronous UDP protocol that communicates servo positions, battery health, and packet acknowledgments. Sam worked on the physical aspect of the project. He designed the layout for the parts on all prototype versions of the vest and installed the various components. Sam also managed the electrical hardware, including wiring of the ADC and Raspberry Pi components. George worked on some non-specialized hardware aspects such as component soldering as well providing assistance with the Unity game engine. In the Unity demonstration, George was responsible for the initial exploration of 3D assets, getting the punching bag model to swing correctly, and understanding the deformable ball scripts that we borrowed for one of our demonstration objects.

Overall, a lack of internal enforcement of maintaining the project development timeline posed the largest threat to meeting the goals of the project. However, the original timeline was quite optimistic, so it's not surprising that it slipped a bit when confronted with the realities of the workloads in our final semester. If we would have had more time with the final hardware prototype, we likely could have dialed things in more to our liking and been more strictly adherent to the desired 100-lb. maximum force capability.

3) Design Documentation

a) Design Concept

The overall hardware design concept for the chest-mounted force feedback hardware system remained largely unchanged from its initial conception in the project proposal. The great variety of components included in the system are elaborated on at length following Figure 1.

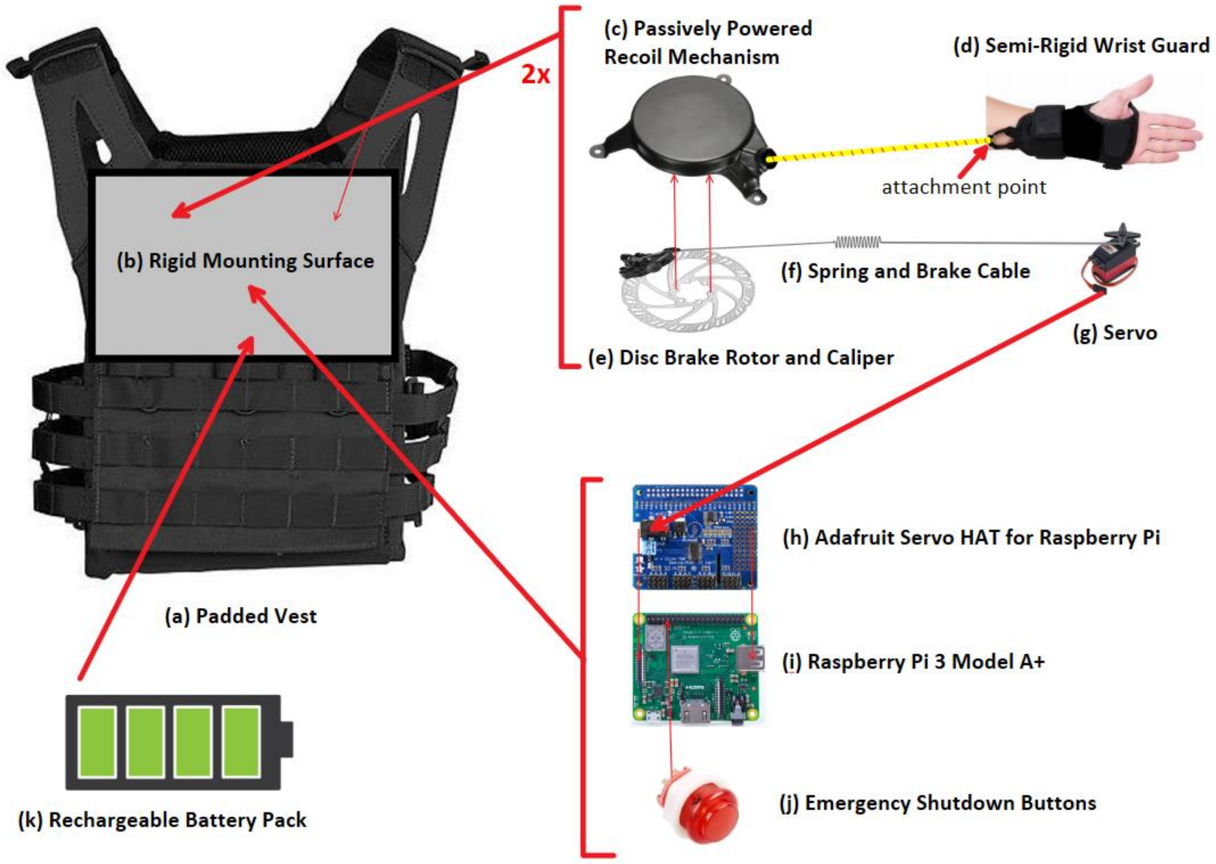

2x
(c) Passively Powered Recoil Mechanism
(d) Semi-Rigid Wrist Guard
attachment point
(b) Rigid Mounting Surface
(f) Spring and Brake Cable
(g) Servo
(e) Disc Brake Rotor and Caliper
(h) Adafruit Servo HAT for Raspberry Pi
(a) Padded Vest
(i) Raspberry Pi 3 Model A+
(j) Emergency Shutdown Buttons
(k) Rechargeable Battery Pack

Figure 1: Chest-Mounted Hardware Overview (product photos from [1][2][3][4][5][6][7]

a) **Padded Vest:**
An adjustable vest securely holds a rigid mounting surface to the chest. Overall, this vest serves as an attachment point for the main hardware components of the system and it comfortably distributes any forces applied to the mounting plate across the torso. The vest itself is derived from modifications made to a bulletproof vest, more technically referred to as a "plate carrier" since it did not come with any protective armor. This vest has plastic buckles above both of the shoulders and around the torso, making it very simple to remove. The buckles allow adjustments in the lengths of the straps so that the user properly adjust the fit of the vest.

b) **Rigid Mounting Surface:**
The mounting surface consists of two separate pieces of HDPE that bolt together to sandwich the exterior fabric of a large pocket on the vest. Hardware components related to the operation of the left and right sides of the force feedback mechanism, the Raspberry Pi, the rechargeable battery, and all other related hardware are securely mounted on this rigid surface. The mounting surface material is 3/8"-thick black HDPE plastic. This was chosen for its electrically insulating characteristics as well as its resistance to cracking.

c) **Passively powered recoil mechanism:**
The nylon cords that attach to the wrist guards originate from a pair of spools mounted to the chest plate. A ribbon spring at the core causes the cable to retract with a light tension

when there is any slack in the cord. This mechanism is derived from an off-the-shelf retractable lawn mower recoil starter.

d) **Semi-Rigid Wrist Guard:**
A pair of semi-rigid wrist guards are secured to the retractable ropes near the base of the wrist guards. Each wrist guard contains a plastic plate that adds rigidity to the inner wrist where the rope attaches. The wrist guards can be snugly tightened with a Velcro strap. Since the wrist is largely immobilized and the attachment point is so far down at the base of the wrist guards, forceful tugging at the attachment point by the force feedback system is not uncomfortable in practice.

e) **Disc Brake Rotor and Caliper:**
A disc brake rotor is bolted into the bottom each of the cable spools. When the arm of the disc brake caliper is actuated by the servo, it is able to resist the outward extension of the rope that is secured to a wrist guard.

f) **Spring and Brake Cable:**
The spring allows for the smooth conversion of servo rotation into tension applied on the brake cable. This is critical to obtaining a smooth gradient of resistances that the system can apply to resist outward arm movement. The spring has the additional benefit of ensuring that the servo will not stall and cause damage to itself.

g) **Servos:**
The disc brake calipers are each be engaged via a brake line and spring by a powerful pair of servos. The model used is a 20.5 kg-cm digital servo with metal gears. These servos engage variable levels of disc brake resistance based on calibration curves that converts a desired resistance to outward movement (a force measured in pounds) into the corresponding servo angle. The servos accept a pulse-width modulated (PWM) signal to control their rotational positioning.

h) **Adafruit Servo HAT for Raspberry Pi:**
The Adafruit servo HAT (Hardware Attached on Top) generates the PWM signal necessary to control the two attached servos. This ensures reliable PWM signal generation and reduces the burden on the CPU of the Raspberry Pi.

i) **Raspberry Pi 3 Model A+:**
The two servos are ultimately controlled by a Raspberry Pi 3 Model A+. The Raspberry Pi uses its included Wi-Fi radio in order to communicate the desktop-based VR software via UDP packets over the local network. The Raspberry Pi runs a Python script that continually monitors for incoming UDP packets that contain instructions on how it is supposed to move the servos that control the disc brakes. Since the servos contain limitations on how they should be used (including a restricted angle range of approximately 135 degrees), this script double checks the adherence of the instructions contained in the received packets to the established limitations and will stop if some erroneous instruction is received. Lastly, the Raspberry Pi interfaces with an external analog-to-digital converter chip in order to read the voltage level of the rechargeable battery. Notifications of low battery levels are sent to the PC so that the user can be notified during gameplay.

The Raspberry Pi runs Raspbian Lite for the operating system since it is optimally compatible with the Raspberry Pi and has large variety of compatible libraries, including Python libraries that we used to interface with the Adafruit servo HAT and the ADC.

j) **Emergency Shutdown Button**
This button is mounted on the exterior of the chest-mounted unit. It is reachable by either hand, even when the nylon cables are fully retracted. When pressed, this button causes the power to disengage from from the servos. The spring tension automatically returns the servos to a neutral position once they are powered off, ensuring that they disengage the disc brakes. After being pressed, the buttons can be rotated in order to have it pop back out to its default electrically closed position

k) **Rechargeable Battery Pack:**
The servos are powered by a rechargeable nickel-metal hydride (NiMH) battery pack rated for 3000 mAh at 7.2V. The Raspberry Pi, the servos, and other electronics on chest-mounted unit are all powered by this single battery. The servos run off of the full 7.2 V while the Raspberry Pi has this voltage stepped down to 5 V by a voltage regulator.

The following components were not visualized in Figure 1, but they are very important to the design:

- **Chest-Mounted Hardware Cover:**

  All the electronics and mechanisms mounted on the chest plate are protected by an acrylic and HDPE cover. This cover protects the user from most of the moving parts and electrical parts of the system and can be removed during maintenance.

- **PC-Based VR Demonstration Program:**

  The force feedback hardware solution is showcased with a demonstration program that is coded in C# using the Unity game engine. This program will demonstrates the ability of the device to apply variable resistance to outward arm movement through the interaction with various objects that your virtual hands can collide with. Additionally, the outer boundaries of the play space will also engage the restriction mechanism in order to demonstrate the potential of the system as a safety device. If the headset falls below a certain height or the acceleration profile resembles a fall, then a safety mechanism will disengage the servos and allow the user to move freely. A similar safety mechanism can also be triggered by pressing a physical button on the chest.

- **Oculus Rift CV1 Headset, Controllers, and Tracking System:**

  The PC virtual reality demonstration program uses the VR play area boundaries provided by the Oculus Guardian System. Of course, this means the VR headset must be an Oculus Rift CV1 headset. The headset is paired with Oculus Touch controllers as well a 3-sensor infrared camera sensor array to provide the precise tracking necessary for the PC software component of the force feedback system. This tracking system is already fully integrated into the Unity game engine by Oculus.

- **Window 10 PC with VR-Capable Hardware:**

  Our PC-Based VR demonstration program was tested on a Windows 10 PC with an NVIDIA GeForce GTX 1080 Ti, a high-end graphics card. We had two such PCs, one from a team member and one provided by Prof. Tyler Bell, the project sponsor.

Modifications to the originally proposed Design:

Despite the relatively high mechanical complexity of the hardware, very few modifications needed to be made to the initial design.

- **Pulleys:**

  One aspect that had been neglected in the original design was the need for a pair of pulleys to redirect the ropes to point straight out from the chest. This positioning is necessary to ensure minimal friction against the rope when performing the most common types of anticipated arm movements. The original concept art did in fact include this sort of rope positioning, but we had simply not thought about a mechanism for making sure the rope can operate this way with minimal friction.

- **Battery Level Detection:**

  We did not specify in the original proposal that we would be monitoring the voltage level of the rechargeable battery. But we decided to implement an analog-to-digital converter that reads the battery voltage through the use of a 2-resistor voltage divider. A UDP packet containing the battery voltage is sent to the PC-based demonstration program periodically so that the user may be notified of a low battery level before it damages the battery or significantly degrades system performance. The ADC chosen is the MCP3008 which features a 10-bit resolution.

- **Type of Vest:**

  We had originally intended to use a padded vest intended for originally manufactured for paintball or airsoft hobbyists. However, we discovered that they sell “plate carrier” vests that are used to carry bulletproof plates up to 11”x14”. This 11” width is exactly the dimension we were targeting with our prototypes, so an 11”-wide pocket on the vest worked perfectly on our final hardware prototype. This choice of vest saved us a great deal of time that would have otherwise involved significant modifications to a less appropriate type of vest. The vest chosen was a NcSTAR Vism ballistic plate carrier vest.

- **Wrist Guard Attachment Mechanism:**

  We had originally thought that it would be best to add a steel cable to the underside of the wrist guard so that the attached rope would have a sliding attachment point that would minimize torque applied to the arm and wrist. However, in practice the wrist guard largely immobilizes the wrist and works very well with a fabric loop attachment farther away from the wrist. In practice, this attachment point works very well and has not caused any discomfort.

b) Standards and Constraints

Constraints:

1. The force feedback must be fully compatible with the Oculus Rift CV1 virtual reality system running on a VR-capable Windows 10 PC:

   We developed the Unity program on a Windows 10 PC with an Oculus Rift CV1 virtual reality system, and we successfully tested a compiled version of this program on a separate Windows 10 PC with a different Oculus Rift headset in Prof. Bell's lab. There does not seem to be any compatibility issues present for this hardware configuration.

2. The end-to-end latency between the virtual reality program issuing a command for full-force resistance and the hardware device applying this resistance level should be less than 0.5 seconds. This constraint is independent of other system requirements.:

   Using a video recording and counting the frames between pressing a button on PC-based GUI that can control the positioning of the servos, we were able to verify that the end-to-end delay of actuating a servo connected to the chest-mounted unit is approximately 0.3 seconds. The packet delay is very insignificant and almost all of the delay is caused by the rating of 0.12 seconds/60° change in servo angle. Even under a significant amount of tension from the spring, the servo appears to have plenty of torque to match its rated rotational speed in a full extended range from 0° to 135°.

3. The total weight of the chest-mounted system, including the vest, must be less than 30 pounds.:

   The final prototype weighs only 18 pounds, well below this limit.

4. The chest-mounted system cannot protrude out more than 6 inches from chest level.:

   All of the components were mounted as flush against the mounting surface as possible so that the cover on the final prototype protrudes no more than 5 inches from the user's chest when including the emergency button.

5. The system must be powered by an appropriate rechargeable battery pack (or system of battery packs). The battery pack must be of a size that can reasonably be integrated into a vest-mounted system.:

   The 7.2-V, 3000-mAh NiMH battery that we selected is approximately 6 inches long and easily accommodated on the chest-mounted hardware system. We did not end up needing a system of batteries since this battery's voltage level did not appear to fluctuate, even when it was under heavy load from both servos. The battery is rated for 30 A, and we never draw anywhere near that much current, thus explaining the lack of voltage drops and spikes.

6. The vest containing the majority of the hardware must be size-adjustable to snugly fit adults of different sizes. There is not a hard requirement on how small or large of an individual it must fit since the design should be modifiable to accommodate an individual of smaller or larger stature, if necessary.:

   The plate carrier vest that we ended up selecting for the final prototype is intended for large-sized adult, but it can be adjusted to a certain extent using the built-in straps.

7. For safety reasons, the vest containing electrical components of the system must be able to be easily removed within 10 seconds.:

   The plastic buckles built into the vest allow it to be very quickly removed by the user. You do not need to slip the vest off in order to remove it. It can be removed incredibly quickly by unclipping the two buckles on the shoulders within a few seconds

8. The chest-mounted hardware must contain a prominent central button (or pair of buttons) that resets the servos to a neutral position (to disengage the disc brake system) and powers down the system. This button should be able to be reached by either one of the user's hands when the cords are fully retracted.:

   We integrated an emergency stop button on the final prototype that was positioned optimally for easy access, even when the ropes are fully retracted and locked. It fully disengages power to the servos.

9. The rigid mounting surface on the chest needs to be backed by an electrically insulating material that is at least one-eighth inch thick to minimize the risk of electrostatic discharge to the user.:

   We selected HDPE for the material of the mounting surface that is closer to the body. This material is electrically insulating, and the 3/8" thickness exceeds the requirement of this constraint.

10. All electrical components and rotating mechanical components must be covered by an enclosure for safety reasons. This enclosure cannot contain any sharp edges that could pose a safety risk to the user during usage. Ventilation holes and slits should be narrow and not be located on the upper part of the enclosure where long hair would otherwise present a safety hazard for the user.:

    The final prototype is covered with a combination of transparent acrylic and black HDPE around the border.

11. The exit points of the two nylon cords on the chest-mounted hardware must be within 8 inches of the center of the nearest shoulder joint (closer is better). This will ensure that the line tension roughly mimics the force vector that an object would naturally exert when the user pushes against it with their hand. This constraint is meant to minimize the risk of user injury.:

    The nylon ropes are plenty close to the shoulders (within 6 inches). We ended up using a pair of pulleys that gave us more freedom in the placement of the rope exit points, so it was very easy to satisfy this constraint.

12. The PC-based VR software and accompanying documentation must make it abundantly clear that the device is a prototype that is absolutely not for use by children or those with relevant health risk factors. Therefore, the VR software must present a disclosure clearly explaining this before the user may proceed in the demonstration program.:

    We display appropriate disclosures in our system of HUD elements.

13. The force feedback system must default to a low-resistance cap of approximately 20 pounds of resistance to outward arm movement. The user may elect to remove this restriction through a further opt-in step that reemphasizes the risks and safety precautions.

This constraint is meant to protect novice users that are not yet used to the force feedback system.:

This feature has not was fully implemented in the menu system.

14. The PC-based VR software should have a warning pop up after 10 minutes of use to remind the user of the possible health risks involved with the overuse of the product. This prototype is intended as more of a technology demonstration, so extended studies would need to be conducted before precautionary messages such as these should be lifted.:

    This notification was also fully implemented.

Standards:

- We will source wireless antennas that comply with FCC Section 15.247 for RF emissions. This regulation pertains to wearable devices to ensure the specific absorption rate of RF signals remains in a safe range.:

  The internal Wi-Fi radio of the Raspberry Pi Model A+ conforms with this standard.

- Our batteries will be sourced from a manufacturer that fulfills ISO 9001 (ensures the organization meets statutory and regulatory requirements related to its products) and ISO 14001 (certifies that the organization has an effective environmental management system).:

  Our selected NiMH battery manufacturer conforms to this environmentally-related standard.

- The end product will make use of network standards for data transmission such as UDP. This will provide us a standard datagram encoding so that the packets can be composed and read by separate programs, even if these programs use a different programming language.:

  Our software solution uses both Python and C# scripts, and they are able to intercommunicate effectively with UDP packets containing standard signed 4-byte integers as well as a 4-byte floating point number.

- Our wireless communication will occur with standard IEEE 802.11ac Wi-Fi transmission frequencies and protocols to simplify implementation and regulatory compliance.:

  The Wi-Fi equipment on the Raspberry Pi and the PC both interoperate perfectly over a 802.11ac-capable router and they all abide by the applicable IEEE standards.

- The Raspberry Pi will use the I2C serial protocol to interface with the Adafruit 16-channel servo HAT.:

  The servo Adafruit servo HAT does use the I2C serial protocol to communicate with the Raspberry Pi over pins.

- The desktop application will use the standard Windows .NET framework baked into C# for interfacing with the Unity engine and the transmission of packets to the Raspberry Pi.

C# scripts written for the Unity game program do leverage the .NET framework for a variety of purposes.

- The Raspberry Pi will very likely run on Raspbian Lite, a well-supported distribution of Linux that is adherent to Linux's standard features.:

  The Raspberry Pi does indeed have Raspbian Lite flashed to its SD card. The Linux command line interface has been very nice to use.

- The software running on the Raspberry Pi will very likely be written entirely in Python and will conform to either Python 2.7 or Python 3.7 programming language conventions.:

  We ended up writing our Python code entirely in Python 3.7 and used libraries compatible with this version of the language.

- Our product will be licensed under the MIT open source license for fair use by whoever wishes to responsibly use or reimplement our designs.:

  We intend to specify this open-source license when the software and relevant documentation are publicly released online.

- This project will not result in a consumer-ready product that will ever be manufactured or shipped to consumers by our team, so we will not need to seek regulatory oversight or compliance for the final protype device.:

  The device is still very much a prototype not intended for mass manufacturing or distribution.

- The SPI standard for communication will be used to interface with the 8-channel analog-to-digital converter (MCP3008). This standard was not originally anticipated to be used since we did not previously think to attach the ADC to read battery voltage levels.

c) Architecture

Full System Integration

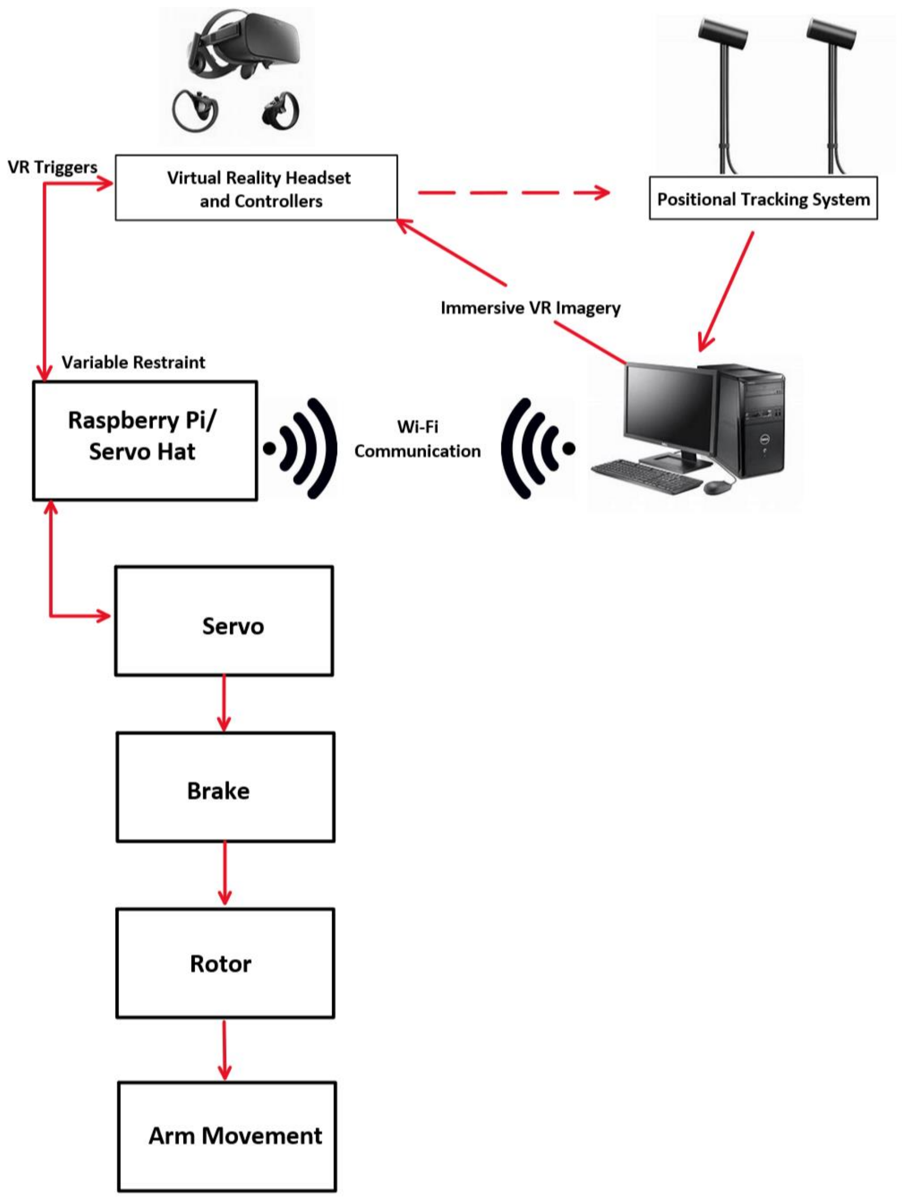


Figure 2: System Overview

The three main components of the system are the wearable force feedback device, a Windows-based PC, and an Oculus Rift VR system. The Oculus Rift includes a fully tracked headset and controllers, courtesy of the IR cameras included in the system. This positional data is fed into the PC that runs a Unity-based VR demonstration program, which is displayed on the Oculus Rift headset. Based on the programming in the demonstration program and the current state of the user relative to their environment, UDP packets are sent via Wi-Fi over the local network to a Raspberry Pi that resides in the wearable force feedback device. These UDP packs contain instructions on how to move the servos. These servos connect to a disc braking system that is able to restrain the outward movement of the left or right arm of the user. This restraining force is facilitated by the tension in a pair of retractable ropes that attach to a pair of wrist guards worn by the user. The forces applied can be very viable, ranging from a passive tension of ~3 pounds to a maximum force of 40+ pounds.

Hardware

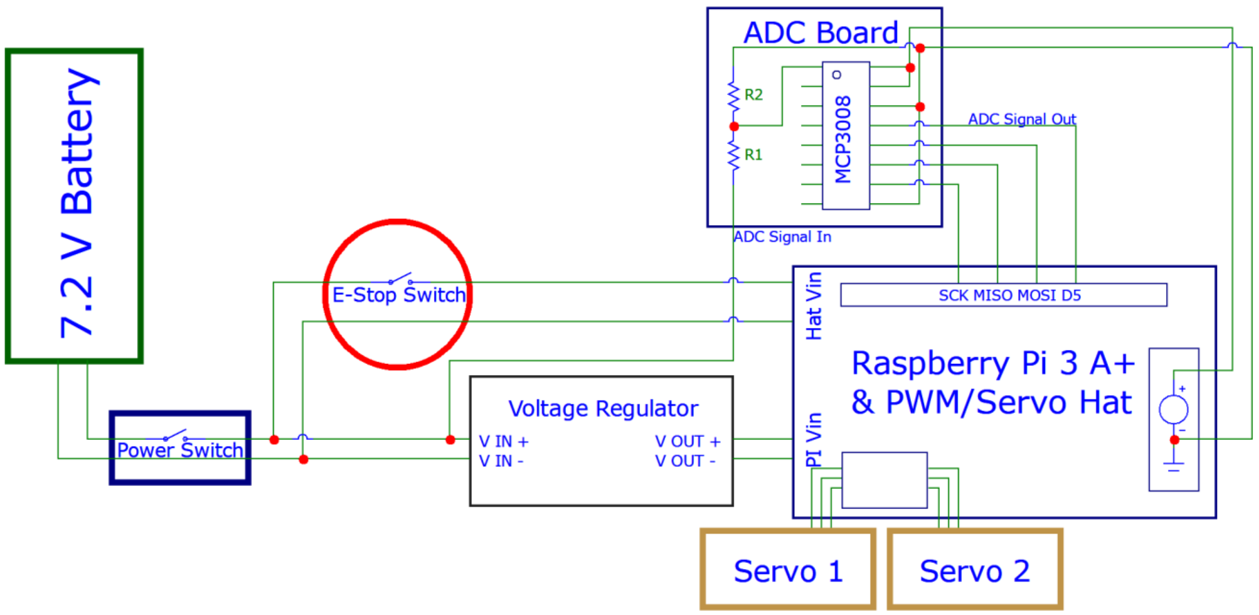


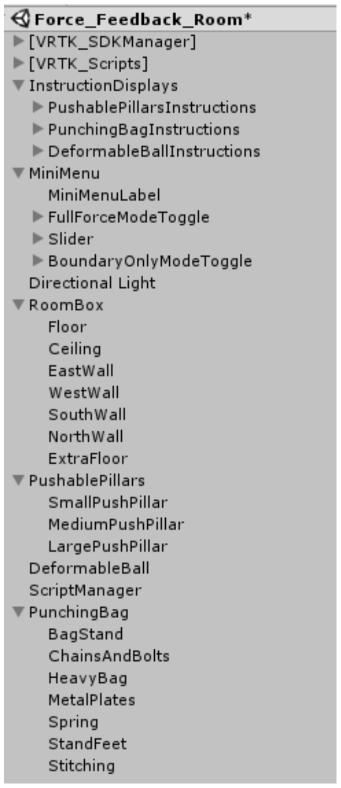
Figure 3: Schematic

The device is powered using a 7.2 V NiMH rechargeable battery with a power switch. 7.2 V goes to three places: the servo HAT through an emergency stop switch, the ADC board for voltage monitoring, and the voltage regulator to be stepped down to 5 V for the Raspberry Pi itself. On the ADC board, a voltage divider is used to bring the 7.2 V to something less than 3.3 V that the MCP3008 IC can use, and this voltage goes to pin 1. R1 is equal to 2.2 MΩ and R2 is equal to 1 MΩ. The board uses four pins on the Raspberry Pi through the HAT, SCK -> pin 10, MISO -> pin 11, MOSI -> pin 12, and D5 -> pin 13. Pins 9 and 14 are connected to ground and pins 15 and 16 are connected to 3.3 V from the Raspberry Pi. The two servos are connected to the servo HAT on the 3-wide male headers. The servo HAT provides the full 7.2 V to these servos along with a PWM signal corresponding to the desired servo angle. Each servo angle effectively corresponds to a certain amount of force to be applied by the force feedback system.

Parts List
NcSTAR Vism Ballistic Plate Carrier Vest
Rigid Mounting Plate for Final Prototype (HDPE)
Cover for Final Hardware Prototype (Acrylic + HDPE)
Wrist Guards (1 pair)
Raspberry Pi 3 Model A+
Adafruit 16-Channel Servo HAT
Samsung EVO Select 64GB MicroSD Card
Recoil Starter Assembly (2x)
160 mm Mechanical Disc Brake Set (1 pair)
High-Voltage Digital Servo (FT5121M) (x2)
Adjustable Switching Voltage Regulator (LM2596)
8-Channel Analog-to-Digital Converter (MCP3008)
7.2V 3000 mAh NiMH Rechargeable Battery
6V-12V Battery Charger
Servo Mounting Bracket (x2)
Raspberry Pi Standoff Kit
Assorted Spring Kit (selected best spring pair)
Steel Bicycle Cables Set
Resistors, Wiring, and Adapters
Misc. Hardware (e.g. screws/bolts)
Luggage Scale (to measure force for calibration)

A full listing of parts is seen above. This list of parts is very similar to what we were originally planning on using, aside from the addition of the ADC and the Acrylic/HDPE casing material.

<u>Software</u>



```
Force_Feedback_Room*
▶ [VRTK_SDKManager]
▶ [VRTK_Scripts]
▼ InstructionDisplays
    ▶ PushablePillarsInstructions
    ▶ PunchingBagInstructions
    ▶ DeformableBallInstructions
▼ MiniMenu
      MiniMenuLabel
    ▶ FullForceModeToggle
    ▶ Slider
    ▶ BoundaryOnlyModeToggle
  Directional Light
▼ RoomBox
      Floor
      Ceiling
      EastWall
      WestWall
      SouthWall
      NorthWall
      ExtraFloor
▼ PushablePillars
      SmallPushPillar
      MediumPushPillar
      LargePushPillar
  DeformableBall
  ScriptManager
▼ PunchingBag
      BagStand
      ChainsAndBolts
      HeavyBag
      MetalPlates
      Spring
      StandFeet
      Stitching
```

<u>Figure 4: Unity Scene Overview</u>

Figure 4 shows the basic components of our Unity-based demonstration program. As can be seen at the top of this listing, we used the Virtual Reality Toolkit (VRTK) in order to facilitate our in-game menu system. This toolkit also allowed us to easily implement an in-game laser pointer that can be used to select menu items and scroll through text. Several of the UI elements are in the form of heads-up displays (HUD). In total, the following UI elements are present in our final demonstration program:

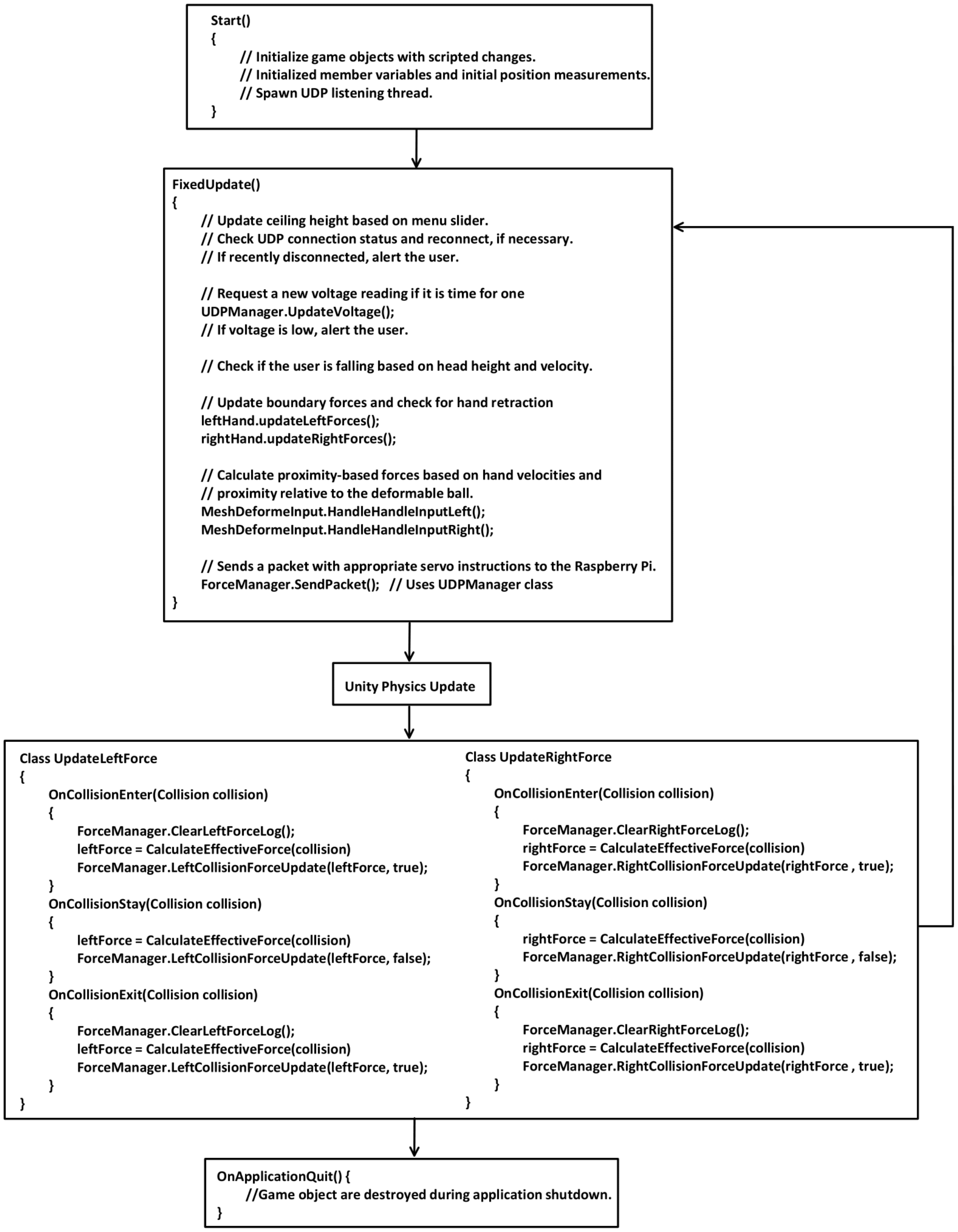
- Initial disclaimer and instructions
- "Mini-menu" system
- Disclaimer about enabling the full-force mode
- Extended use warning HUD (after 10 minutes of use)
- Low voltage alert (based on battery level of wearable force feedback system)
- Raspberry Pi connection/disconnection alerts
- Instructional signs in front of all of the demonstration stations

If the source code was opened up in Unity, these HUD elements would be listed under:
VRTK_SDKManager -> VRTK_SDKSetups -> Oculus -> OVRPlayerController -> OVRCameraRig -> TrackingSpace -> CenterEyeAnchor

The scripts relevant to the operation of the VR demonstration program are contained in Assets -> Scripts. **Since the source code .zip file is quite large and this location isn't necessarily easy to find, we have included all relevant scripts that we wrote in Relevant_Scripts.zip**. This file contains the Python script that runs on the Raspberry Pi as well as those the C# scripts for Unity that were directly written by us (i.e. not part of a library, etc.).

Figure 5 on the next page gives a good overview of the functionality we have implemented in the Unity demonstration program. Unity's physics engine updates on a fixed interval set by Time.fixedDeltaTime. By default, this means the physics are update once every 0.02 seconds (i.e. 50 Hz). Since the physics are updated in a non-continuous fashion, it is actually impulses that are applied to object, not forces. However, you can calculate the equivalent force by dividing by Time.fixedDeltaTime. This is how we calculate the amount of force that occurs during object collision in Unity. Since all of the units in Unity are in terms of standard metric units, the forces are in Newtons. However, for convenience, our system converts these forces to pounds. As you can see in the Figure 5, the built-in functions OnCollisionEnter(), OnCollisionStay, and OnCollisionExit() allow us to make decisions based on the current state of a collision. Both the left and right hands have been given spherical colliders that will trigger these functions upon intersection with another collider. There is a complex series of dot products, calculations, and logic that occur within the ForceManager class's functions. The particularities of this class are best deduced from the source code comments since it becomes quite involved. During both the OnCollisionEnter() and OnCollisionExit() functions we pass "true" as a second argument to indicate that the force feedback hardware system should be required to immediately update the servo's physical angle. This is intended to reduce any lag in the system. However, during a continued impact we pass "false" to indicate that the servo angle should only be updated every so often and only when the difference in force is significant enough to warrant it. This decision partially based on running average of the last 5 force measurements which are stored in a queue (i.e. the last 0.1

seconds).



```
Start()
{
      // Initialize game objects with scripted changes.
      // Initialized member variables and initial position measurements.
      // Spawn UDP listening thread.
}
```

```
FixedUpdate()
{
      // Update ceiling height based on menu slider.
      // Check UDP connection status and reconnect, if necessary.
      // If recently disconnected, alert the user.

      // Request a new voltage reading if it is time for one
      UDPManager.UpdateVoltage();
      // If voltage is low, alert the user.

      // Check if the user is falling based on head height and velocity.

      // Update boundary forces and check for hand retraction
      leftHand.updateLeftForces();
      rightHand.updateRightForces();

      // Calculate proximity-based forces based on hand velocities and
      // proximity relative to the deformable ball.
      MeshDeformeInput.HandleHandleInputLeft();
      MeshDeformeInput.HandleHandleInputRight();

      // Sends a packet with appropriate servo instructions to the Raspberry Pi.
      ForceManager.SendPacket();   // Uses UDPManager class
}
```

```
Unity Physics Update
```

```
Class UpdateLeftForce
{
      OnCollisionEnter(Collision collision)
      {
            ForceManager.ClearLeftForceLog();
            leftForce = CalculateEffectiveForce(collision)
            ForceManager.LeftCollisionForceUpdate(leftForce, true);
      }
      OnCollisionStay(Collision collision)
      {
            leftForce = CalculateEffectiveForce(collision)
            ForceManager.LeftCollisionForceUpdate(leftForce, false);
      }
      OnCollisionExit(Collision collision)
      {
            ForceManager.ClearLeftForceLog();
            leftForce = CalculateEffectiveForce(collision)
            ForceManager.LeftCollisionForceUpdate(leftForce, true);
      }
}
```

```
Class UpdateRightForce
{
      OnCollisionEnter(Collision collision)
      {
            ForceManager.ClearRightForceLog();
            rightForce = CalculateEffectiveForce(collision)
            ForceManager.RightCollisionForceUpdate(rightForce , true);
      }
      OnCollisionStay(Collision collision)
      {
            rightForce = CalculateEffectiveForce(collision)
            ForceManager.RightCollisionForceUpdate(rightForce , false);
      }
      OnCollisionExit(Collision collision)
      {
            ForceManager.ClearRightForceLog();
            rightForce = CalculateEffectiveForce(collision)
            ForceManager.RightCollisionForceUpdate(rightForce , true);
      }
}
```

```
OnApplicationQuit() {
      //Game object are destroyed during application shutdown.
}
```

Figure 5: Unity Game Software Flow

In contrast to the collision forces just discussed, boundary forces are the forces that we apply to prevent the user from extending their hands outside of their play space. These forces do not depend Unity's physics calculations, although they are updated at the same 50 Hz frequency. Instead, these forces are determined by the state of the user and the positioning of their hands and headset. Based on the positioning of the headset, a pair of shoulder objects are placed close to where the user's actual shoulders would be located when standing upright. These shoulder objects are used to understand the relative direction of a hand movement (i.e. inward, outward, etc.) and to aid in various calculations. With Oculus's tracking system API, we are able to calculate the proximity of the nearest boundary wall to any object, including the hands of the user. There are 3 situations when a boundary force activates. First, if the hands come within 15 cm of an outer boundary wall (including the ceiling), the servos will activate with full force to prevent any attempt to push a hand outside of the play area. Second, when a hand is within 60 cm of an outer boundary, the servo moves to an angle that is just below the threshold where the resistance would become noticeable (there is a lot of "dead space" in the servo angle range before significant resistance is applied to the brakes). This gets the servos ready to very quickly activate a full force restraint if the user moves in a way that indicates that could exit the play space within a certain threshold time (currently set to 0.3 seconds). This is based on the current velocity of the hand as well as the positioning of the shoulder (i.e. they need to be within reach of the boundary). Once the software predicts that the hand could exit the play space at a relatively fast speed, it activates a full-force resistance for at least a set amount of time (currently set at 0.5 seconds).

In addition to the force feedback that ensures that the user stays within their play space boundaries, we have also implemented vibrational feedback in the controllers. As the user's hand comes closer to a boundary, the vibrational motor in the Oculus Touch controller will vibrate faster and with a higher intensity. Since our force feedback system has ~0.3 seconds of mechanical lag and other interactions that cause force feedback, this vibrational haptic feedback is meant to provide the user earlier tactile feedback that alerts them to their proximity to the edge of their play space.

A MeshDeformerInput class facilitates a thirds type of force that we calculate. This script calculates a proximity-based force that grows stronger as the user pushes their hands toward our deformable ball game object. This object is a large sphere hovering in the air that reacts to hand movement towards it once the hand is within 1 meter of its surface. We use the velocity vector of each of the hands in order to project a "SphereCast" that will tell us if the current hand trajectory would collide with the deformable ball mesh within 1 meter. If a SphereCast collision is detected, we apply a deforming force that is directly related to the speed of the hand movement and the proximity to the deformable sphere. Fast hand movements towards the ball within close proximity to its surface will cause the greatest deformation of its elastic surface. Although this deformation is dependent on hand speed, the proximity-based force that we calculate and send to the force feedback system hardware is based solely on proximity to the surface of the deformable ball. This ensure that this force does not change very rapidly, allowing the servos to keep up with the changes and provide an experience that is less sensitive to the mechanical lag of our system (up to ~0.3 seconds).

Since collision forces, boundary forces, proximity-based forces can all occur simultaneously, we always calculate a "dominant" force that is the largest of the three (unless the boundary-only mode has 2 of these forces turned off). However, the three sets of forces do behave differently. Changes to the servo angles resulting from a collision are only "forced" to be implemented by the Raspberry Pi immediately during a collision entering and a collision exiting event. However, to increase the responsiveness of the boundary forces system and proximity-based forces, every single servo change resulting from a boundary force or proximity-based force is “forced” to be executed immediately.

In order to keep all of the actions occurring easier to follow, the functionality has been divided among a small number of scripts. The central coordinator script is ScriptManager.cs. In addition to the already discussed functionality, it also responds to menu system choices by the user (such as changing the ceiling height), calculates if the user is falling (which would trigger a disengagement of the force feedback system), requests voltage reading updates, and turns on/off various HUD elements. The workhorse behind most of the force calculations is ForceManager.cs. This is the most complicated script by a fair margin. The bidirectional UDP communication is handled by UDPManager.cs. The aforementioned proximity-based forces are updated by MeshDeformerInput.cs. LeftHand.cs and RightHand.cs are attached to their respective in-game objects in order to calculate forces relevant to their current collisions and positioning. Finally, pushable_pillar.cs is a simple script that sets spring constants for the “pushable pillar” in-game objects. These spring constants are proportional to the volume and mass of these pillar (all set to an equal density).

The Raspberry Pi script that directly controls the servos, servo_controller.py, is also contained in the Relevant_Script.zip file. This script creates a UDP listening socket in order to process and acknowledge any instructional packets it receives from the PC-based demonstration program. It uses an ADC to measure its battery voltage level whenever requested to do so. It leverages Adafruit’s ServoKit library in order to correctly configure the servo HAT to control our servos throughout they effective 135° range.

The UDP messaging protocol format consists of four values for all packets between the Pi and the PC. The first 3 of these values are signed 4-byte integers. The last value is a float. The normal mode of operation is “command mode” which is used to send servo angels to the Raspberry Pi. The first 2 values of the message are integers that represent the angles the left and right servos should be set between 0 and 135 degrees. The third integer indicates if these servo angle changes are “forced” and should therefore circumvent some the regular checks that keep the servo operation consistent. Lastly, there is a time stamp stored in a float that indicates the times since the start of the Unity game. This time stamp only increments of unity of Time.fixedTimeDelta = 0.02 seconds, so this acts as a packet numbering system.

The other mode is “health query” that can be used by setting the third integer in the packet to 4. When this is done, no angles are changed and the Pi quickly responds with the voltage reading of the battery. Some examples of the message format and uses are shown below.

Command mode

Messaging protocol Examples			
Angle of Left Servo	Angle of Right Servo	Servo Change Forced?	Float: Time stamp from Unity
0	0	0: Neither servo needs to “force” this angle change immediately	156.54
90	0	1: Only the left servo needs	156.56

		to “force” this angle change immediately	
0	87	2: Only the right servo needs to “force” this angle change immediately	156.58
90	87	3: Both servos need to “force” this angle change immediately	156.60

<u>Device Health Query</u>
*Servo Angles if set are not changed when check value is 4

Messaging protocol Example				
Angle of Left Servo (not used)	Angle of Right Servo (not used)	Message Type	Float: Voltage reading from Pi battery	
-	-	4	-	Send To Pi
-	-	4	7.11	Receive From Pi

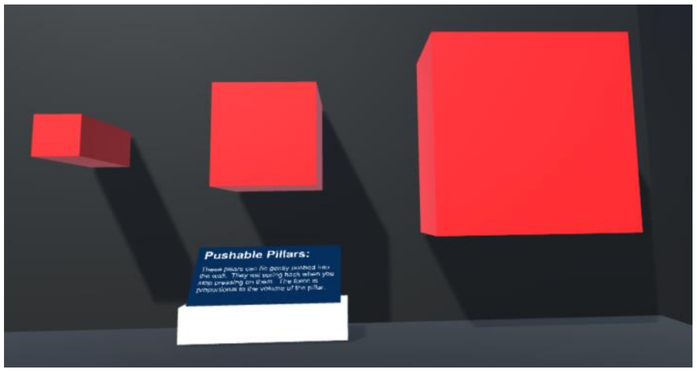
d) User Interface

The user interface (UI) for the Unity demonstration is made up of game objects that the user can interact with. These interactions can take place at one of our 3 stations: the punching bag, the pushable pillars, and the deformable ball. The user makes use of the Oculus Touch controllers to interact with these demonstrations with their “hands” being visible in-game. The 3D environment is rendered to the user and displayed on the Oculus Rift headset. These objects are show from the user’s perspective in Figures 6 through 8.

Punching Bag:
Please don't strike the punching bag rapidly. Give the system time to respond with slower interactions.

<u>Figure 6: Punching Bag</u>

The user’s interactions with the punching bag are relatively intuitive. The user is able to punch the bag and fell the force feedback from each of these collisions. However, as they are in formed in the instructional display, they should not attempt to strike it too rapidly for the best experience. Rapidly punching the bag will cause the servos to be unable to keep up. This is simply due to the ~0.3 seconds of mechanical lag that occurs during a 0 to 135° change in the servo angle. However, the interactions are quite fun as long as the user exercises some mild restraint.

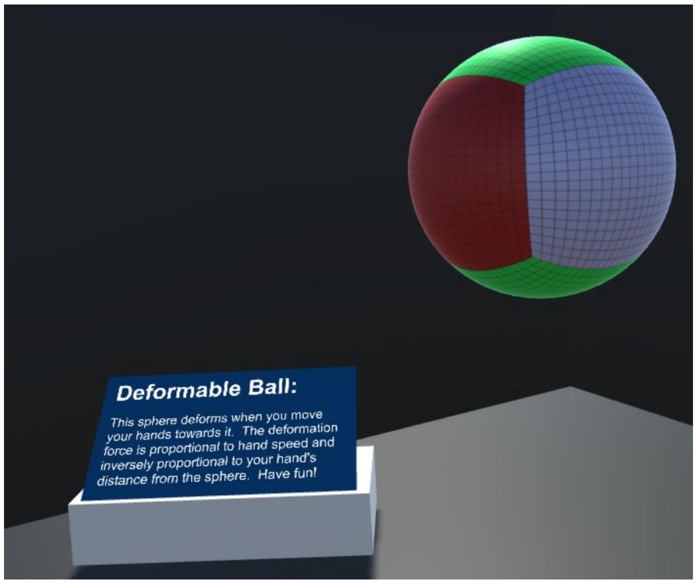


Pushable Pillars:

These pillars can be gently pushed into the wall. They will spring back when you stop pressing on them. The force is proportional to the volume of the pillar.

Figure 7: Pushable Pillars

The pushable pillars demonstration lets the user experience a smoother force curve with a gentler mode of interaction. These forces are based on the mass of the pillars (i.e. inertia) and the spring tension pushing them back towards their starting positions. Since there are 3 different sizes of pillars, the user is able to experience the full range of forces possible with our force feedback system.



Deformable Ball:

This sphere deforms when you move your hands towards it. The deformation force is proportional to hand speed and inversely proportional to your hand's distance from the sphere. Have fun!

Figure 8: Deformable Ball

The deformable ball is a very fun demonstration where a user can interact with a ball. In this demonstration, the user will feel a force that is inversely proportional to the distance from the user's hands to the deformable ball. When a user is interacting with the ball, it will deform based on these interactions. Although the forces that the user experiences are only based on proximity, the deformation of the ball itself is sensitive to hand speed and velocity, opening the door to a lot of fun ways to play with it. Leaving the ball to sit for a few seconds will allow the ball to come back to a resting state. This deformable ball was based on an online tutorial [8] which we modified in order to be fully compatible with VR (it was originally just a click-based interaction; not nearly as fun or interactive).

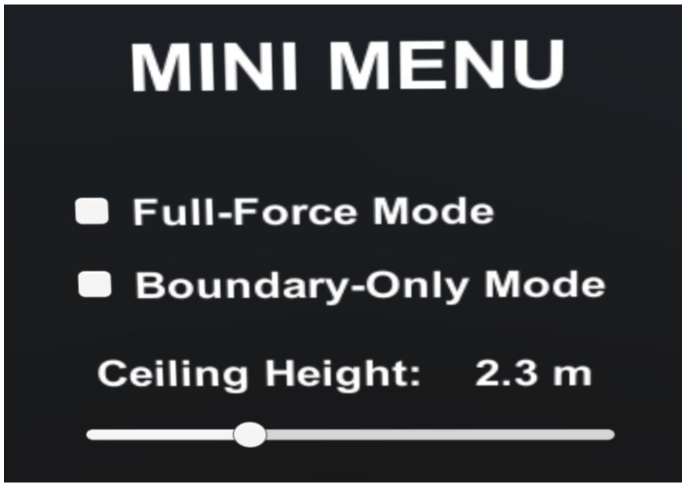


Figure 9: In-Game Menu System

The “mini menu” is placed in the demonstration room on one of the walls. This menu can be controlled by projecting a green laser from your hand using the Oculus touch controllers. This laser can be accessed by pressing and holding either joystick and then pressing the rear trigger button on the controller to make your selections. Enabling Full-Force Mode will enable the full resistance of the force feedback device, whereas disabling it will only give a max resistance of 20 lbs. By default, the Full-Force Mode is disabled. Enabling the Boundary-Only Mode will disable all forces that are not related to keeping the user’s hands inside of their safe play space. This boundary is displayed to the user as a mesh surface as they approach it. The Boundary-Only Mode is intended to reduce strain on users that elect to use it.

e) *Maintenance*

- The NiMH battery will need to be routinely charged via an external charger. An in-game low battery notification will notify you of the need to recharge the battery. Additionally, the cover of the chest-mounted hardware can be removed so that you can easily read the voltage level of the battery pack from the voltage regulators readout. It is recommended that the battery be recharged once it drops below 7.0 V.
- After several hundred charge/discharge cycles, the NiMH battery pack will need to be replaced. This need will be made evident when the battery pack does not last as required.
- Occasionally, the pulleys underneath cover of the chest-mounted hardware system will need to be lubricated for optimal performance and a reduction in wear on the part. We recommend a silicone spay lubricant.
- If the nylon cords ever begin to fray, they should be replaced or the damaged section should be cut off. Replacing the cord is not difficult once the recoil mechanism has been unbolted from the mounting plate.
- For optimal system performance, always make sure the PC and Raspberry Pi are both connected to a high-performance, relatively uncongested Wi-Fi router. It is highly inadvisable to use a public Wi-Fi router. Additionally, the 5 GHz Wi-Fi band is

strongly suggested due to the relative lack of interference and congestion at that frequency.

- The hardware system should be routinely calibrated. A C# GUI is provided that allows you to manually specify a servo angle within the range of the servo. After engaging the servo to a specific angle, you can measure the line tension that this servo angle is able to exert in the wrist-attached cables. Once a series of calibration measurements have been performed that cover a wide range of servo angles, these data can be to the relevant trio of member variable in the ForceManager class of the Unity game project. These member variables are the following: calibrationForces, leftForceCalibrationAngles, and rightForceCalibrationAngles. You must perform measurements corresponding to the same set of calibrated forces, but you are able to collect as many measurements as desired. The software will linearly interpolate values based on the provided calibration data.
- It is very possible to modify the chest-mounted hardware system to better suit your needs. For instance, if you can find a slightly stiffer pair springs to use, you may be able to obtain a more rapid response than the current system. Additionally, there are likely other forms of braking mechanisms that may have more favorable braking characteristics, specifically hydraulic brakes that are often favored over the mechanical brakes used in this system.

4) Test Report

<u>Physical Hardware System Tests</u>

Criteria	**Conducted Test**	**Results**
Cable retraction mechanism must ensure that slack in the cable never interferes with the user or device.	Starting at initial position test each cable individually by pulling cord to max length. Then quickly move end of cable to initial position and ensure slack doesn't interfere	Pass
The force feedback system must be able to resist at least 100 pounds of force applied to the retractable cords on each side of the device.	Use a luggage scale the measure the maximum force that the force feedback system can resist without releasing any length of rope.	Fail: The device was only capable of 40 lbs. in its final calibrated state. The braking mechanism is capable of more, but it would require additional tweaking.
For safety reasons, the vest must be able to be easily removed within 10 seconds.	With the Oculus Rift on, time how long it takes the user to remove the vest. Verify this time is under 10 seconds.	Pass: The vest can be removed within 3 seconds when using the buckles.
The total weight of the chest-mounted system, including the vest, must be less than 30	Weighed the final prototype with a luggage scale.	Pass: Final weight = 18 lbs.

pounds.		
The chest-mounted system cannot protrude out more than 6 inches from chest level.	Measured the depth of the chest-mounted hardware.	Pass: Hardware is 5" deep when including the protruding emergency button
The chest-mounted hardware must contain a prominent central button that resets the servos to a neutral position.	Checked that the emergency button disengages servo power.	Pass

Unity Program Tests

Criteria	**Conducted Test**	**Results**
The system must activate the vibrational motors in the Oculus Rift controllers as a complementary alert mechanism to the boundary restraint of the force feedback device.	Without the force feedback device on validate that the Oculus Rift controllers vibrate when not in the play area. Finally start from the center of the play area and walk to each boundary and validate that the controller's vibration increases as the user gets closer to the boundary.	Pass
The system must have a togglable "boundary control"-only mode that intervenes only when the user is moving your hands outside of the safe play space (and is not activated by interactions with in-game objects).	When "boundary mode" is enabled a user's interactions with in-game objects will not engage the servos. The servos only engage when the user's hands are exiting or outside the play boundary.	Pass
The PC-based VR software should have a warning pop up after 10 minutes of use to remind the user of the possible health risks involved with the overuse of the product.	The user should be reminded after 10 minutes of use that they should take a break to lessen the risk of injury.	Pass
The force feedback system must default to a low-resistance cap of approximately 20 pounds of resistance to outward arm movement.	On game start-up, the menu setting full-force mode should be unchecked and a maximum force of 20 lbs. should be possible.	Pass

Raspberry Pi Software Tests

Criteria	Conducted Test	Results
Ensure that the Pi can control the servos independently and, on the servos, full range of motion.	Use python test program to validate that the library allows use of the full range of motion (0 - 135) and that the servos operate independently.	Pass
Ensure that the Pi can send and receive UDP packets from PC over a local network.	Set up a C# console app to send UDP packets to the Pi and verify that the Pi is receiving these messages by sending a copy of the message back to the PC as an acknowledgment.	Pass

Full System Tests

Criteria	Conducted Test	Results
The force feedback system should be able to prevent the user's hands from exiting the play area when an arm movement begins with the hand initially 2 feet or more within the outer boundaries of the VR play area.	With the play area physically mapped out on the floor first validate that a user standing in the center of the play area is not being restricted. Second, have user approach the boundary wall and punch and validate the user is not restricted but that the servo has engaged to 70 degrees to prepare for a hard stop. Last, have the user go to the edge of the mapped play area and punch again and validate the servo is fully engaged and the user is restricted.	Passed with flying colors
The end-to-end latency between the virtual reality program issuing a command for full force resistance and the hardware device applying this resistance level should be less than 0.5 seconds.	Use a GUI tool to change a servo angle form 0 degrees to 135 degrees while connected the chest-mounted hardware. Record this so that the time between setting the 135 degree angle with the GUI tool and the servo reaching the correct positioning can be measured by counting frames in the video.	Pass: ~0.30 seconds from 0° to a maximum angle of 135°

5) Appendices

A. References

[1] Amazon, “Poweka Recoil Starter Pull Start Assembly.” [Online]. Available: https://www.amazon.com/Poweka-Assembly-751-10299-951-10299-951-10299A/dp/B00M9DEXO0/ref=pd_day0_hl_86_7?_encoding=UTF8&pd_rd_i=B00M9DEXO0&pd_rd_r=57e2f6cf-264d-11e9-ad19-fb6cb727d522&pd_rd_w=uUuic&pd_rd_wg=Xa130&pf_rd_p=ad07871c-e646-4161-82c7-5ed0d4c85b07&pf_rd_r=RSKT9R5M0NSA0EK0MP8P&psc=1&refRID=RSKT9R5M0NSA0EK0MP8P. [Accessed: 04-Feb-2019].

[2] Amazon, “Tektro Mountain Bike MD-M280 Disc Brake Caliper 160mm Rotor.” [Online]. Available: https://www.amazon.com/gp/product/B01MXK1EGO/ref=oh_aui_search_asin_title?ie=UTF8&psc=1. [Accessed: 02-Apr-2019].

[3] Pololu, “FEETECH High-Torque, High-Voltage Digital Servo FT5121M.” [Online]. Available: https://www.pololu.com/product/3427. [Accessed: 04-Feb-2019].

[4] Amazon, “CTHOPE Wrist Guard, Impact Protective Glove Wrist Brace Support Pads for Snowboarding, Skating, Skiing, Motocross, Mountain Biking.” [Online]. Available: https://www.amazon.com/CTHOPE-Protective-Snowboarding-Motocross-Mountain/dp/B079DS4CXK/ref=sr_1_73?ie=UTF8&qid=1549312994&sr=8-73&keywords=wrist%2Bguard&th=1. [Accessed: 04-Feb-2019].

[5] Raspberry Pi Foundation, “Raspberry Pi 3 Model A+.” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-a-plus/. [Accessed: 04-Feb-2019].

[6] Amazon, “EG STARTS 5 Piece 24mm Full Color LED Illuminated Push button Built-in Switch 5V Buttons.” [Online]. Available: https://www.amazon.com/d/Electrical-Light-Switches/EG-STARTS-Illuminated-Joystick-Raspberry/B01LXZSV2N/ref=sr_1_14_a_it?ie=UTF8&qid=1549316729&sr=8-14&keywords=raspberry+pi+button. [Accessed: 04-Feb-2019].

[7] Discount Paintball, “Valken Plate Carrier II Paintball Vest Black.” [Online]. Available: https://www.discountpaintball.com/Valken-Plate-Carrier-II-Paintball-Vest-Black_p_10589.html. [Accessed: 04-Feb-2019].

[8] Catlike Coding “Mesh Deformation: Making a Stress Ball.” [Online]. Available: https://catlikecoding.com/unity/tutorials/mesh-deformation/. [Accessed: 05-May-2019].