Hand Force Controller

A force sensing gaming and computer controller

A Proposal Submitted by Team 9

Lillian Boettcher

Grant Kveton

Dhaval Manojkumar Shirvi

Affiliations

Florida Atlantic University

Department of Engineering and Computer Science

Course: ENG 4952C - 001

Instructor: Dr. Hari Kalva

Submitted 6 August 2020

Project Summary

Accessibility to computer functionality, gaming, and virtual reality is becoming more demanding as technology advances further and more people are using these devices and technologies. Many devices exist for controlling computer and game commands, most popularly a computer keyboard and mouse as well as certain specialized gaming controllers. For virtual reality, controllers are moved in real life and translate to movements in a virtual space using internal accelerometers and gyroscopes. These have limitations for some individuals, primarily in how they often require excessive movement. There are some controllers made for people who have difficulty moving their hands or limbs effectively enough to use these devices, but they normally consist of additional buttons and still require a large level of movement.

The Hand Force Controller is a proposed device to solve this problem by using force sensors to read subtle amounts of force in a user's fingertips and throughout their hand. The force needed to use the controller can be specified through user calibration. Additionally, the sensors can be moved around to however the user finds them most comfortable. Therefore, the Hand Force Controller is a stationary controller that can be specialized to how the user wants the layout and how much force they want to apply in order to use it. The controller comes with an application that is straightforward to use, a 3D printable platform that can be customized to the user's preferences, and a user manual. The software application has quick calibration settings and a simple interface for accessing other functionality. To demonstrate how it can be used in gaming and virtual reality, there are two game demos, a first-person shooter game and a playable virtual piano. As part of the main functionality, there is a program for setting different controller gestures to different keyboard commands so the controller can emulate any computer game.

Thus, the Hand Force Controller is a combination of hardware and software to present a specialized gaming controller and keyboard emulator.

(Boettcher)

Table of Contents

Introduction

Background

Controlling computer software originally uses a simple keyboard and mouse to maneuver selecting system objects and typing documents and commands. However, with an increase in the complexity of computer gaming and virtual reality (VR), specialized computer controllers are now often used for different games with different playstyles. These controllers require the user's hands to be in different positions and perform different positions. Many controllers use large movements of joysticks or the controllers themselves through motion sensing, which can allow for more immersive experiences or more possible ways to control game actions.

These controllers vary greatly and are continuously in demand as the number of people who play games and perform various computerized tasks increases and as controller and game technology advances. These controllers are mainly geared towards people who are passionate towards gaming and VR and want more immersive or specialized control in their games or simulations. However, some controllers exist for people with disabilities, such as a loss of limbs or trouble with movement, which allows people with physical disorders to control computer commands and participate in gaming.
(Boettcher)

Problem Statement

While there are some gaming controllers for people who have physical disabilities, they are primarily geared towards people who have amputated limbs or distorted figures. There are few controllers for people who have trouble with simple movements, such as rapidly moving their fingers up and down, which is required on nearly all types of gaming devices. For virtual reality in particular, there is little in the way of controllers that do not require large limb movements, since VR uses movements in real life to translate to movements in a virtual environment. Because of this, controllers do not largely exist for VR for disabled people who cannot move well enough to use VR controllers and equipment. Therefore, while there are people who need special devices for playing computer games without having to perform excessive movements, especially for VR simulations, there are few controllers that have proven effective enough to be a solution for these individuals.

(Boettcher)

Scope of Work

Overview

1. Design Stage

The Hand Force Controller begins with an initial design stage that involves researching what components will be needed to accomplish the goals of the initial idea. The design stage proved to be an iterative process as the team finalized the system requirements in the original proposal document and determined what components would be best suited for the project. The initial hardware design involves developing circuit diagrams based on research, so the components can be quickly assembled and tested when received. Software libraries and tools are researched in this stage to determine a practical design and implementation of the desired software applications to be delivered with the controller. The list of hardware components is finalized for the project budget and confirmed to be ordered.
(Kveton)

2. Hardware Implementation Stage

The hardware implementation stage begins once the hardware components have been ordered and received. The hardware implementation takes a bottom-up approach in that the complexity of the project increases as more components are implemented and tested. The first stage involves configuring a single sensor and having values read by the Arduino microcontroller. The second stage involves configuring multiple sensors to be read by the Arduino. The final stage of the hardware implementation occurs when the number of sensors

exceeds the number of analog inputs available on the Arduino microcontroller. A multiplexer is implemented in this final stage to allow for a greater number of sensors to be used.
(Kveton)

3. Software Implementation Stage

The first stage of the software implementation is the programming and testing of serial communication between the Arduino microcontroller and the computer running the applications created in Unity. Similar to the hardware implementation, this stage takes a bottom-up approach by iteratively introducing and testing new components. First, the communication between devices is tested without using any sensors. Once the communication is established, the system for transmitting the sensor data can be implemented. The other software applications in the project are heavily dependent on this component. Therefore, testing various use cases and handling errors is very important.

The second stage of the software implementation involves programming the three demo simulations that are used to demonstrate the Hand Force Controller's capabilities. These three applications have been divided up between the team such that each team member has an application to work on. The keyboard command simulation and the piano simulation were decided upon in the early design stages for the project. The idea to include a demo video game was decided upon later in the project during progress update discussions with the professor. Another aspect of this stage in the software implementation is the sensor calibration system. This is designed to be implemented in a way such that it can be used in the demo applications.
(Kveton)

4. Final Interface Implementation and Testing

Once each of the software subsystems have been implemented and tested on their own, they are connected together by an interface within one Unity project. This stage also involves final testing of the entire system with all of the subsystems connected to the main screen interface. The development testing of each subsystem on their own should allow for simplified integration into the main system interface.
(Kveton)

Literature Review

Team Member Reviewing: Lillian Boettcher

Title: “The Human Controller: Usability and Accessibility in Video Game Interfaces”

This paper focuses on analyzing current availability of accessibility in gaming controllers and why there is a need for an increase in accessibility. It addresses an initial fact that there are concerns, such as cost concerns, related to providing accessibility and having all controllers require a large amount of accessibility is not practical or feasible. Despite this, it points out that there is an existing demand for accessible controllers and that there are people who develop disorders, including gamers, as they age or go through life, covering aspects of the demand. It evaluates case studies to determine possible strategies for going about creating accessibility in places where it is lacking.

The ideas on how to improve upon current lacking technologies for the physically impaired gives insight into specific tasks our project should accomplish. More specifically, it stresses that there is little in the way of easy to use controllers for the physically disabled. These constraints would need to be met by creating a device that requires minimal movement and difficult performing any required movements, which is the key focus of this project.

Team Member Reviewing: Lillian Boettcher

Title: "Game Not Over: Accessibility Issues in Video Games"

This paper covers the scope of which accessibility for games is in demand. It covers both physical and mental aspects of where games may lack accessibility, from not providing subtitles to follow along to audio to not being able to use the controllers that a game console requires. These issues stem from disabilities in four different categories: visual disabilities, auditory disabilities, mobility disabilities, and cognitive disabilities.

The Hand Force Controller relates to the third addressed disability category: mobility disabilities. It supports the need for accessibility controllers by relaying statistics on how 12.1% percent of the population has some sort of mobility disability. It also addresses that there are individuals who may play games and want to continue after receiving these abilities from aging or injury, the most common causes of these types of disorders, further proving the presence that this is a problem with a demand for a solution.

Team Member Reviewing: Grant Kveton

Title: HandNavigator: "Hands-on Interaction for Desktop Virtual Reality"

This paper presents an innovative interactive system for controlling digital models using hand gestures. The physical interaction device is utilizing force sensors to measure finger and palm gestures which are then translated into software actions, similar to the Hand Force

Controller. The physical interaction device uses forces exerted from the palm and fingers to control a virtual hand model, rather than measuring real hand movements and gestures. The paper discusses the necessity of implementing plausible real-time interactions with the virtual hand to enhance the user's immersion. This is done by correctly mapping the physical device to a well designed virtual hand model. Visual feedback, when done properly, can give the user the perception that they are actually touching virtual objects.

Team Member Reviewing: Grant Kveton

Title: "Game Console Controller Interface for People with Disability"

This paper proposes an interface device to allow people with upper limb disabilities to play console video games. Currently, there are few devices that address the difficulties disabled users encounter when attempting to play video games. As video games have become more complex, the more challenging it has become for users with disabilities to play using traditional controllers. The device introduced in the paper seeks to implement an entirely new interface device that uses off-the-shelf sensors and switches that can be used by disabled users to play video games on a traditional console. The paper also describes the implementation of a method for the user to customize the mapping of sensors to joypad pushbuttons, a similar feature to the Hand Force Controller's custom keyboard input subsystem. Another similar system requirement to the Hand Force Controller is that the interface device must allow a simple setup procedure, allowing the user to utilize the device with various applications.

Team Member Reviewing: Dhaval Manojkumar Shirvi

Title: "Evolution of Game Controllers: Toward the Support of Gamers with Physical Disabilities"

This paper focuses on the limited consideration for players with physical disability throughout the evolution of game controllers. It covers the disadvantage the gamers with physical disability such as limited mobility, dexterity or a missing limb or part of the limb have when using the standard controllers. It also covers some of the modified controllers that reduces the disadvantage on the physically disabled players. However, the modified controllers are made considering the players with a missing limb.

The Hand Force Controller answers the issue for the players with limited mobility or dexterity. The controller was designed considering the players with limited mobility. And since, the controller does not require hand movement except for exerting slight pressure, it could be a potential solution for players with limited mobility or dexterity.

Alternative Solutions

There are many alternative approaches to VR controllers but there are not many that require minimal movement. Currently many VR technologies use touch sensors in their controllers. Most touch sensors use digital output, which is high if touched or low if not. However, some touch sensors have the capability to output analog value. Even so, the VR touch sensor controllers require hand movement. Another alternative to force sensors could be flex sensors. The flex sensors measure the deflection or bending, which can be one of the most effective sensors for the immersive VR experience. The flex sensor might be a good approach for immersive VR controllers, but it still requires hand movement. Although, it might not require as much movement as touch sensors.

One of the most important hardware components is the microcontroller board. A microcontroller board reads the values from the sensors and sends it to the computer. There are many Arduino boards that have the same functionality, such as Arduino Leonardo and Arduino micro. Arduino Leonardo and micro are also compatible with the keyboard library of the Arduino. Arduino Leonardo and micro boards come with 12 analog pins (Arduino). However, both boards are a bit advanced for beginners.

For the software aspect, creating an application for the user to test the functionality of the controller, can be achieved on almost any platform. For example, the unreal engine by epic games and using python executable directly. The unreal engine uses C++ scripts for the applications, which has a method to simulate a key press. The unreal engine also provides high quality visuals (Unreal Engine). Although the keyboard simulation only works on Windows OS and unreal engine requires a high powered PC to run the applications due to the high-quality visuals. On the other end python has the keyboard library and can also be used to create an application for the user (Python). However, the application would not be very user friendly.

(Shirvi)

Evaluation

The main concept behind the Hand Force Controller is to minimize the hand movement of the user while performing in-game actions. The force sensors have great potential to achieve the goal, since it only requires for the user to exert a small amount of pressure. The other sensors such as touch and the flex sensors require some hand movement to perform actions. Therefore, the force sensor is a more preferable choice, since it requires the least amount of movement of all the three sensors. The Arduino Uno is the base model of some of the other Arduino boards. Arduino uno in comparison to Arduino Leonardo and Arduino micro, has less functionality but Arduino Uno is more beginner friendly than the other two boards. Even though Unity might not have high quality visuals or a keyboard simulation functionality, it would be the best choice for making a user friendly application which can be run on a low end PC. And to achieve the keyboard simulation functionality, ironpython can be used to execute python commands from cs script. Therefore, unity makes an ideal choice for this project.
(Shirvi)

Decision

Many physically challenged could benefit from a new type of controller with another level in how easy it is to use. To make an effective controller that allows individuals who are physically disabled, the main factor requirements the controller needs is ease of use by minimizing required movement. This could be done by using force sensors to detect small and weak amounts of pressure in a user's hand, which could be customly placed around an individual's hand. Additionally, to maximize accessibility, it should be held in an open-source repository that developers and users can easily access, have an adjustable platform and calibration, and have clear instructions and examples on how to use it.

(Boettcher)

Problem Implementation Plan

Research

The Arduino programming documentation was researched during the early stages of design and implementation to design software functionality necessary for meeting the functional requirements (Arduino). The analogRead() function allows the Arduino sketch to read values from the analog force sensors connected to the analog input pins. The Serial object contains functions related to the implementation of the serial communication, such as the println() function and the readStringUntil() function.

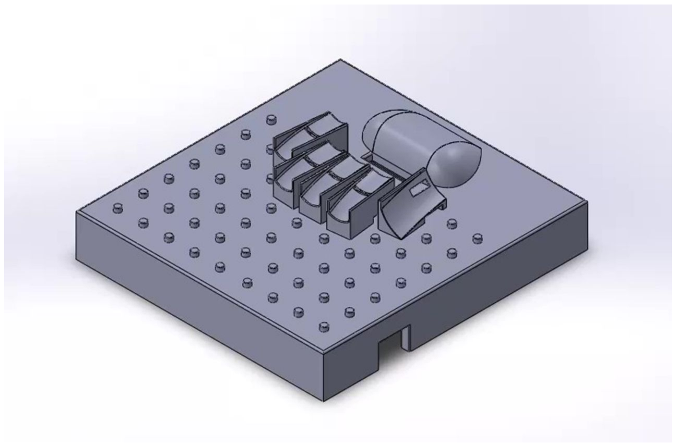
The official C# documentation from Microsoft was researched during the early stages of design and implementation to design software functionality for serial communication in Unity (Microsoft). Specifically, the SerialPort class in the System.IO.Ports namespace was researched. Similar to the Arduino Serial object, this class includes functions for opening the serial port connection, reading from the serial port buffer, and writing to the serial port buffer. Code examples in the documentation were studied for understanding of the SerialPort class but were not used in the implementation of the project.
(Kveton)

Design

The initial design of the project included an Arduino Uno for utilizing five force sensors, one for each finger. The design was later changed to incorporate three more force sensors. Since the Arduino Uno only has six analog input pins, a multiplexer chip was added for the extra sensors to connect to the Arduino. This design increases the ability of the Hand Force Controller to take into account a majority of basic hand movements. The first five force sensors are for each finger and the extra three force sensors are for the palm, the left palm, the right palm and the back of the hand. These sensors are to be placed on an adjustable platform comfortably position the sensors around the user's hand.

The platform design involves three components: the finger and thumb components, which are used for resting the finger and thumb while using the controller, and a base. Each finger rests on four finger components and the thumb component is used for the thumb. As the finger and thumb components are used for resting fingers and thumb, the base platform can be used to rest the palm of the user's hand. The base platform has space underneath it, for placing the Arduino Uno, and has space for force sensors to rest on the finger and thumb components. The finger and thumb components also have slits for the force sensor wires to pass through and be placed on top of the components so that the wires can be out of the way of the user's fingers.

Image 1: Platform Model with Combined Components



The Hand Force Controller can be tested on its application, which includes calibration settings, a piano simulation, a keyboard emulator, and a first person shooter game. In the piano

simulation, the Hand Force Controller can be used to play piano in a top view. The piano simulation can be converted to VR simulation by connecting a VR headset to the camera display. The keyboard emulator enables the Hand Force Controller to emulate keyboard keys and commands. This allows the user to use the Hand Force Controller on any application. The first person shooter game can be used to test the Hand Force Controller's ability to play a common style of computer game.
(Shirvi)

Deliverables

The project deliverables include:

- The Application
    - Calibration
    - The Piano Simulation
    - The Keyboard Emulator
    - First Person Shooter game
- Printable 3D modeled platform
    - Base platform
    - Finger component
    - Thumb component
- Instruction manual
- Repository for the deliverables
- Hardware design

The application includes the piano simulation, the keyboard emulator and the first person shooter that demonstrates different functionalities of the Hand Force Controller. The 3D modeled platform includes three components one is the base platform, one is finger and the last one is thumb. The instruction manual is a detailed step by step guide for assembling the hardware and the 3D modeled platform. The manual also includes a guide for using the application. The repository is for development purposes and to download the application, printable 3D platform and the instruction manual. The hardware components include all the hardware components required for assembling the Hand Force Controller.
(Shirvi)

Tasks

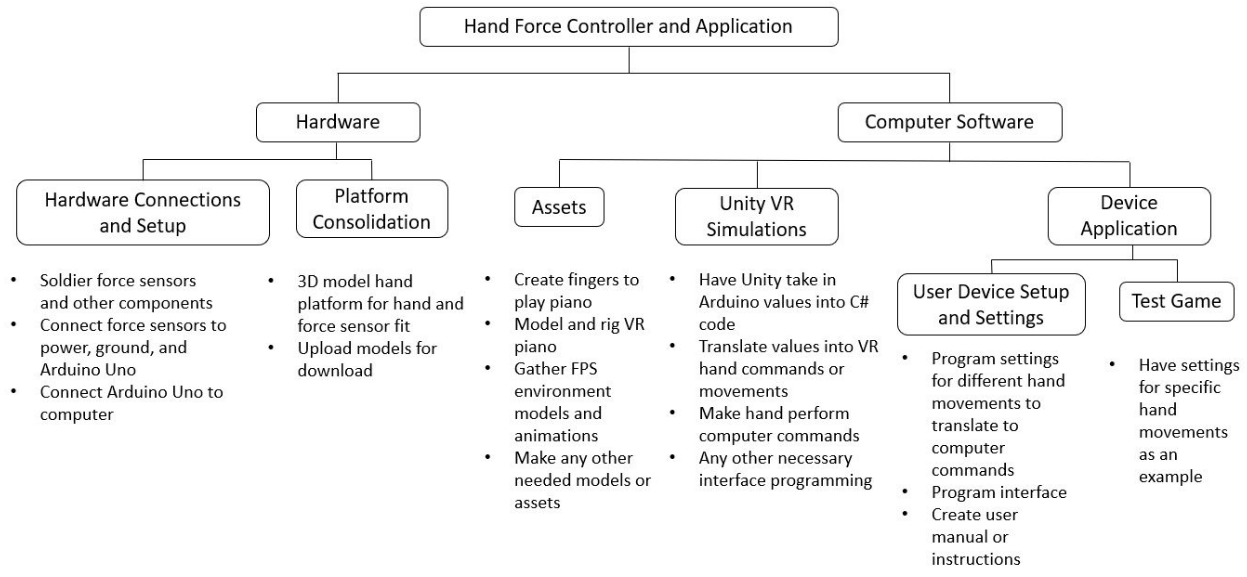
The project was given tasks that are designed to finish the project with all of the functional and nonfunctional requirements. The simplified task division is listed below in a diagram. These tasks were originally made to allow us to divide the work between group members and calculate how much work we have done and how much we had left. The following is a list of tasks and the team member leading in completing the task:

- Gather all hardware components (all)
- Design hardware (all)
- Manage an accessible repository for the application, code, and development materials (Kveton)
- Create a user manual that gives instructions on how to use the Hand Force Controller and replicate our work (Boettcher)

- Model components for a 3D printable platform (Shirvi)
- Main application:
  - Create a main application display (Boettcher)
  - Create calibration system (Boettcher)
  - Create navigation to other parts of software (Boettcher)
  - Consolidate all parts of the application and deploy as executable (Kveton)
- Keyboard emulator (Shirvi):
  - Create a UI for emulation settings (Shirvi)
  - Create script for saving emulation settings on play (Shirvi)
  - Program active emulation script in Python using saved settings (Shirvi)
  - Program Python receiving of sensor values from Arduino (Kveton)
- Piano simulation (Boettcher):
  - Model a 3D piano asset (Boettcher)
  - Connect audio files to piano keys (Boettcher)
  - Script piano playing functionality based on saved calibration (Boettcher)
- First-Person Shooter Game (Kveton):
  - Gather assets to form game scene (Kveton)
  - Program player movement and actions based on calibration and HFController settings (Kveton)
  - Program enemy intelligence (Kveton)
  - Program health and ammo systems (Kveton)

(Boettcher)

Figure 1: Task Division Diagram


Hand Force Controller and Application
Hardware
Computer Software
Hardware Connections and Setup
Platform Consolidation
Assets
Unity VR Simulations
Device Application
User Device Setup and Settings
Test Game
• Soldier force sensors and other components
• Connect force sensors to power, ground, and Arduino Uno
• Connect Arduino Uno to computer
• 3D model hand platform for hand and force sensor fit
• Upload models for download
• Create fingers to play piano
• Model and rig VR piano
• Gather FPS environment models and animations
• Make any other needed models or assets
• Have Unity take in Arduino values into C# code
• Translate values into VR hand commands or movements
• Make hand perform computer commands
• Any other necessary interface programming
• Program settings for different hand movements to translate to computer commands
• Program interface
• Create user manual or instructions
• Have settings for specific hand movements as an example

Prototype Construction

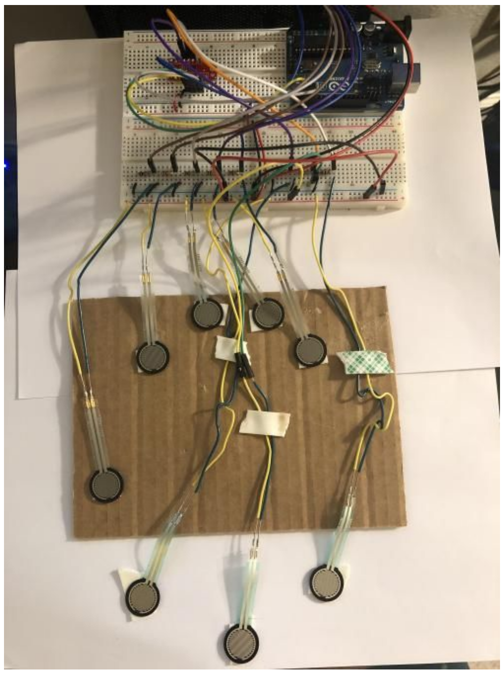
The complete initial prototype consists of eight force sensors to portray the complete functionality. However, the 3D modeled platform is not a necessity for playing the simulations created, calibration, or using the Hand Force Controller as a keyboard emulator. Therefore, the platform was not printed and used for the initial prototype.

The initial prototype uses a running Arduino with a USB cable to print to a computer. The model could be made using another form of connection, however, such as a Bluetooth adapter. All eight force sensors are connected to the five-volt power supply from the Arduino and a ground. Before the ground, however, there is a resistor for each sensor in order to allow the sensors to read the pins. The initial prototype uses 10 kilo ohm resistors for the specific sensors used in the prototype, but the design allows for other types of force sensors, which may require different level resistors. In order to increase the analog inputs in the Arduino Uno, a multiplexer is used. This is gone into more detail in the hardware section.

The prototype had no issues in terms of calibration, lag, or other common problems known with prototypes. For future modifications in improving the Hand Force Controller, more force sensors can be added for more control over software and allowing more possible gestures. In addition, more platform models could be made to be more effective for different individuals and possibly easier assembly. Parts could also be more specialized; the jumpers used in the prototype make it different to consolidate the hardware and better hardware components could probably be found.

(Boettcher)

Image 2: Hardware Prototype



Implementation Details

System Specifications and Functionality

The original functional and non-functional requirements were all generally met at the conclusion of the project. There were no requirements officially added to the project, but the project came to have more of a focus on also demonstrating more capabilities of the Hand Force Controller, meaning the first non-functional requirement became more significant as demonstrating the controller's use became an important part of the project.

Functional Requirements

1. The system shall measure the force exerted by each finger of a human hand as an increasing or decreasing analog value.
2. The system shall quickly and accurately transfer analog data from an Arduino microcontroller to a computer.
3. The system shall be able to translate different analog inputs into different computer keyboard commands.
4. The system shall be able to recognize different finger force patterns to translate into commands or actions.
5. The system shall be implemented in such a way that there are minimal dependencies required during setup when using the controller for different applications.

6. The physical platform for the controller shall include design and functionality that accounts for varying hand sizes between users.

Nonfunctional Requirements

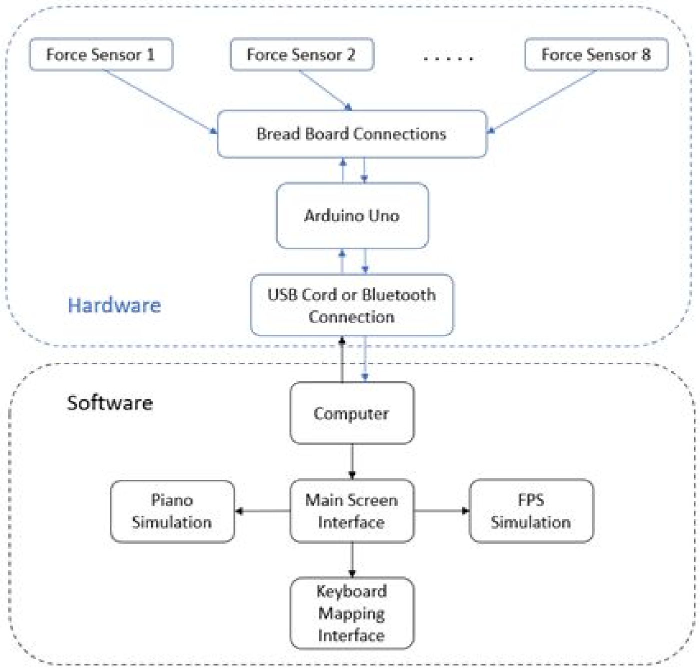
1. The product shall include an application for users to test the product.
2. The product shall include a manual or guideline for users.

(Boettcher, Kveton)

System Design

The overall system can be divided into two categories: a hardware subsystem and a software system. The hardware system forms the Arduino Uno, force sensors, and their hardware connections. The software consists of a Github repository containing a user manual, 3D model platform components for 3D printing, and an application. The Hand Force Controller application can be divided into four parts: the main screen, which is where calibration and scene navigation takes place, a keyboard emulator, a piano simulation, and a first-person shooter game. (Boettcher)

Figure 2: Overall System Design Diagram



Hardware Subsystem Design

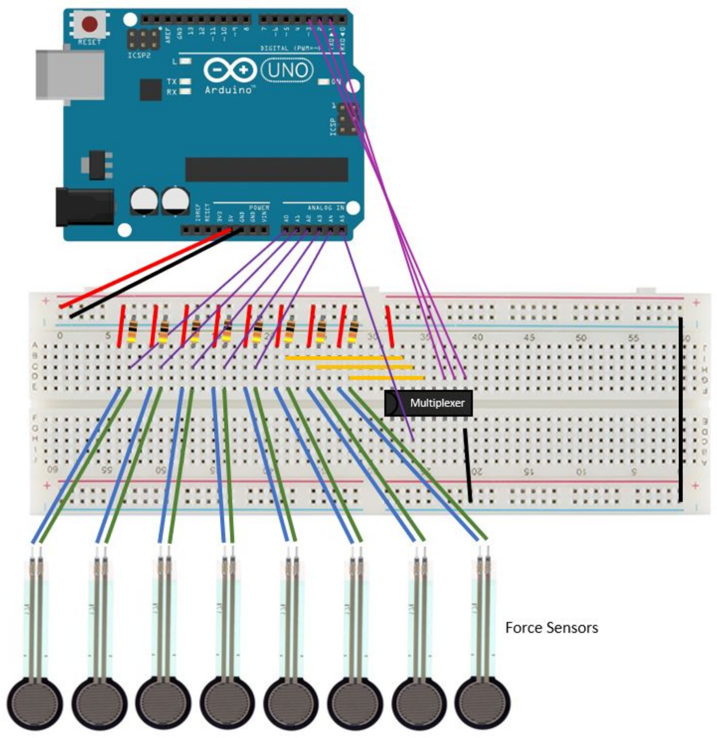
The microcontroller used in this project is the Arduino Uno Rev3 model. The project uses all six analog inputs for the purpose of reading sensor input data. Five analog inputs are directly connected to force sensors. The remaining analog input is connected to the common output pin of the multiplexer. Three of the digital pins are connected to the select pins of the multiplexer and are set to output values for determining which individual input pin of the multiplexer is to be read from. The Arduino microcontroller has a USB type B port where the Arduino is connected to a PC via USB cable, and this port is the connection in which the serial data communication occurs.

Force Sensitive Resistors (FSR) are thin sensors that detect physical pressure upon the sensor. As one applies pressure to the force sensor the resistance goes down and a measurement is given based on the change in resistance (Adafruit). The range of values for the force sensors in this project was from 0 to about 1000, with 0 representing no force applied and 1000 representing a large amount of force applied.

Due to the limitations of the Arduino's amount of analog input pins, it was necessary to implement a multiplexer into the hardware design once the project began using more than six force sensors. The multiplexer used in this project is the 74HC4051 eight-channel multiplexer breakout. A multiplexer is able to iterate over its individual input pins by using three "select" pins to determine which pin's data will be output. The "select" pins are connected to the digital output pins of the Arduino. The multiplexer has 8 individual input pins, and these are selected by outputting their corresponding binary values to the "select" pins. For example, if the input pin *Y1* is to be selected, then the digital pins would output "*LOW, LOW, HIGH*" which could also be interpreted as the binary value *001* . The multiplexer has a common output pin that is connected to an analog input pin of the Arduino where the currently selected input's data is sent (SparkFun Electronics).

(Kveton)

Figure 3: Circuit Diagram



Serial Communication Subsystem Design

The Arduino microcontroller writes data being read from the force sensors to a serial port buffer. A C# class in the Unity game engine, called HFController, then reads the incoming data from the serial port buffer. This system was first implemented and tested by writing dummy data to the serial port to be read by the HFController class on the receiving computer. Once the communication had been established and tested, the sensor data could be used in communication. The sensor readings for each loop within the Arduino code were packaged in a data structure of comma separated values. The packets of readings were then written to the serial port buffer. The packet data structure required a parsing algorithm to be designed within the C# code to unpack the string of comma separated values to be stored in a local array. A public method for the

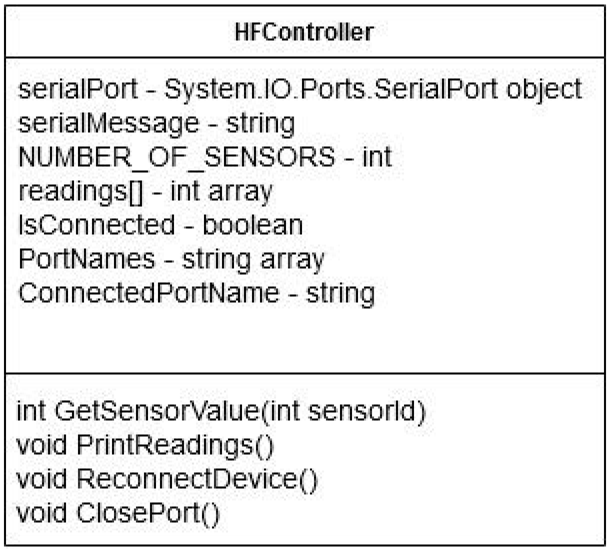
HFController class, GetSensorValue(int sensorId), was developed to return the current reading for a given sensor number.

In the early stages of development, the port name was hardcoded into the Unity C# code. An issue with this implementation is that various machines may have differently named serial ports when the Arduino is connected to the PC. An algorithm was developed to iterate between all available ports on the computer, sending a "wake up" message to the port. The Arduino code is configured to wait for the "wake up" message and will send a response message when it is received. When the C# code receives the response, it will keep that port open for communication. There is also a timeout mechanism in place so that the Arduino will go back to its sleep state, awaiting the "wake up" message, if it does not receive a "reading" message from the serial port after the timeout value has passed. In the later stages of development, a public method for the HFController class, ClosePort(), was needed to close ports as the main interface switched between demo applications. The HFController class and the Arduino sketch are programmed such that only the NUM_OF_SENSORS constant needs to be changed when adding more sensors to the system.

The HFController C# class is designed in such a way that it abstracts the serial communication and can be used modularly within various applications. The figure below shows the fields, properties, and methods for the HFController class. The class is also documented in greater detail in the readme of the project's GitHub repository.

(Kveton)

Figure 4: HFController Class Variables and Functions



HFController

serialPort - System.IO.Ports.SerialPort object
serialMessage - string
NUMBER_OF_SENSORS - int
readings[] - int array
IsConnected - boolean
PortNames - string array
ConnectedPortName - string

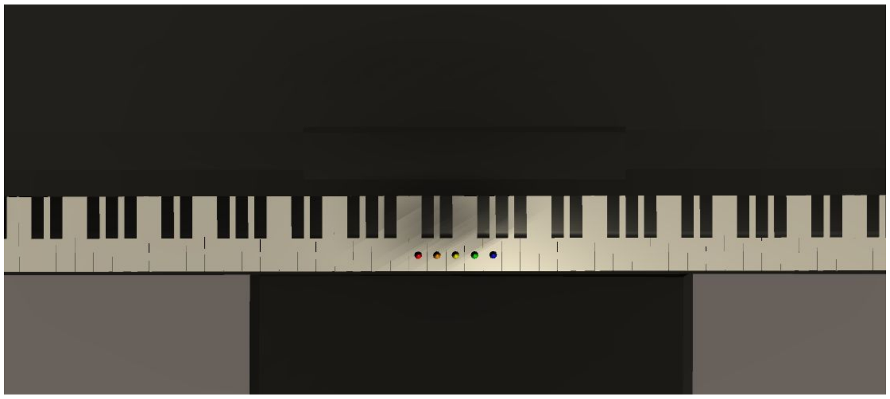
int GetSensorValue(int sensorId)
void PrintReadings()
void ReconnectDevice()
void ClosePort()

Piano Simulation Subsystem Design

The piano simulation is designed to demonstrate the functionality of the Hand Force Controller to work for playing a computer simulation or virtual reality game. The game can simply be played in a first-person view, with the camera at an orthogonal top view, but the camera view can be adjusted to be controlled in respect to a VR headset, turning it into a VR simulation.

The simulation is played using all eight of the force sensors. The first five force sensors (sensors 0-4) correspond to the first five fingers of the human hand. The controller only has a single hand and uses differently colored capsule models to represent fingers that can press keys, so the user can use their right or left hand to control the virtual fingers. Therefore, the fingers are referred to by number and color instead of the name of the finger, with finger 1 colored red, finger 2 colored orange, finger 3 colored yellow, finger 4 colored green, and finger 5 colored blue.

Image 3: Piano Simulation View



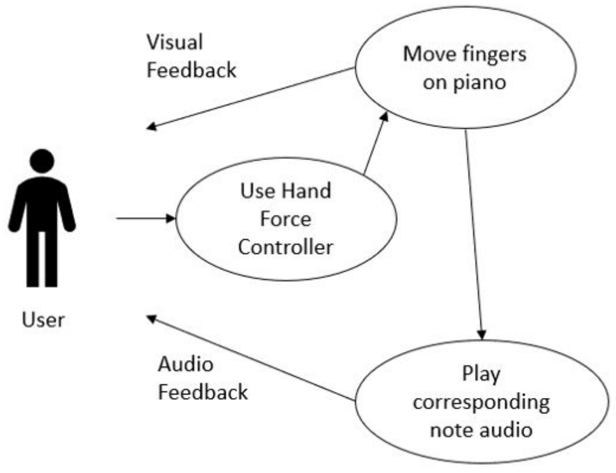
Force sensors 5 and 6 control the left and right movement of the fingers, respectively. The last force sensor, sensor 7, is used to move the fingers forward towards the black keys on the piano. Keys are pressed by pressing the first five force sensors. The amount of force needed to press the notes for each finger is defined by the calibration in the previous main screen. This also applies to the amount of force to activate left, right, and forward movement for the remaining three sensors.

Key pressing is recognized by the location of each finger. For each finger, the simulation programming checks for the position of the finger. The x and z coordinates give the position of the keys, which are saved in array variables. These coordinates are checked for each finger, which defines what key can be pressed. The value of the finger force sensors define the y coordinate position of the fingers. If the finger is lowered to a certain degree, the note at its position is activated. An activated note changes color and plays its corresponding audio file. A white note will change to pink while activate and black notes to purple. When the position of the finger changes to a different note through a change in the x or z coordinates or when the y coordinates are raised above a certain threshold, the note audio that was previously playing stops.

The x coordinates of all of the fingers, as well as the camera, are changed using sensors 5 and 6. When sensor 5 is pressed to a value above its calibration value, each finger and the camera move left, or across the negative x-axis, a specific degree each frame update. The amount is defined by the degree to which the sensor value is greater than the calibration value. The same applies to force sensor 6, except it creates right movement when pressed across the positive x-axis.

Force sensor 7 moves all of the fingers forward on the z-axis, which allows the black keys to be playable. When the sensor value is above the calibration value for sensor 7, the fingers all move forward. The amount moved per frame is, like with the left and right movements, defined by the amount the sensor value is larger than the calibration value. When the sensor value is lower than the calibration value, the fingers move backwards towards the white key only region (where black keys cannot be reached or played along the x-axis). (Boettcher)

Figure 5: Piano Simulation Use-Case Diagram

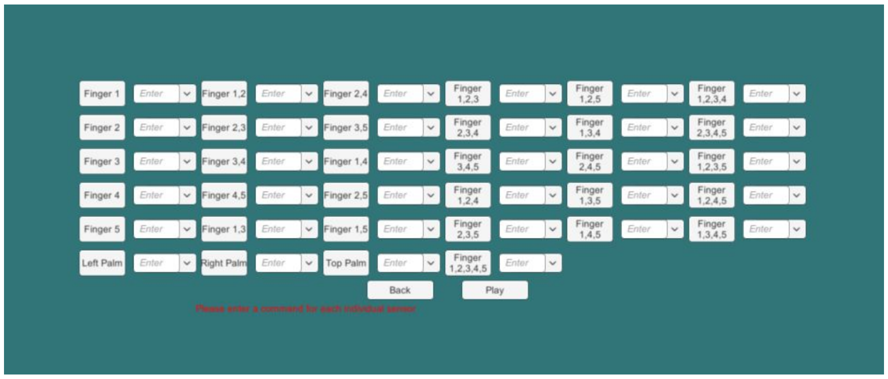


Keyboard Emulation Subsystem Design

The keyboard emulation program is designed to turn the Hand Force Controller into an emulator for any computer application. The main purpose of the keyboard emulation subsystem is to be able use the Hand Force Controller to play different video games, though it can be used on other applications as well.

The keyboard emulation program uses different gestures that can be performed on the Hand Force Controller to emulate different keyboard commands. The gestures include the individual eight force sensors and combinations of the first five force sensors. All the gestures are available to be viewed on the keyboard command screen. The gestures are simple and self explanatory. Eight of the available gestures correspond to the eight force sensors. The combination of the first five force sensors (for each finger) can be used for combinational gestures. A key or a command can be assigned to each available gesture on the keyboard command screen. It is mandatory to assign keys to the eight individual gestures, however assigning a key to the combinational gestures is optional. A play button can be pressed when the user is done assigning the keys or gestures.

Image 4: Keyboard Emulation Settings

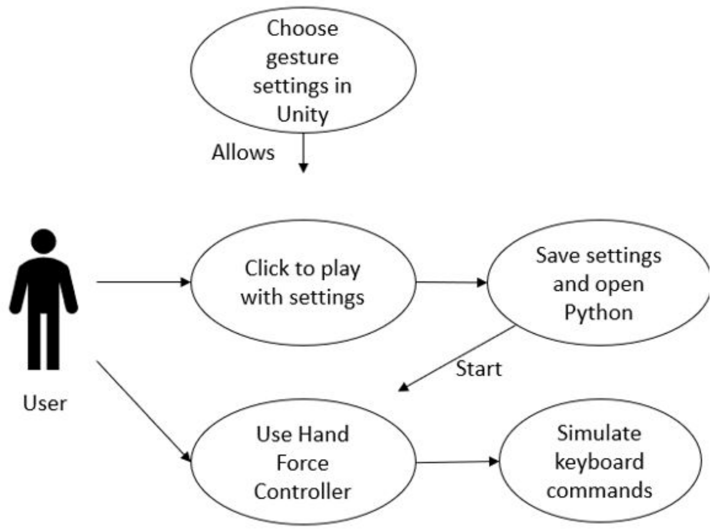


Please enter a command for each individual sensor

Once the play button is clicked, the assigned keys and the port name are saved to a text file and a Python script is executed using IronPython. IronPython allows C sharp scripts to execute or use Python commands. The Python script reads the keys and the port name from the text file using an open() function. The port name is then used to connect to the Arduino via a serial library. The Python script also uses the keyboard library to simulate key pressing. The keys read from the file are used by the keyboard.press() function, which performs key presses. The values of the force sensors are constantly being read in a main loop from the Arduino by the Python serial library. Every time one of the gestures is performed on the controller and the reading is above a certain threshold, based on saved calibration values, the key assigned to each respective gesture is pressed.

(Shirvi)

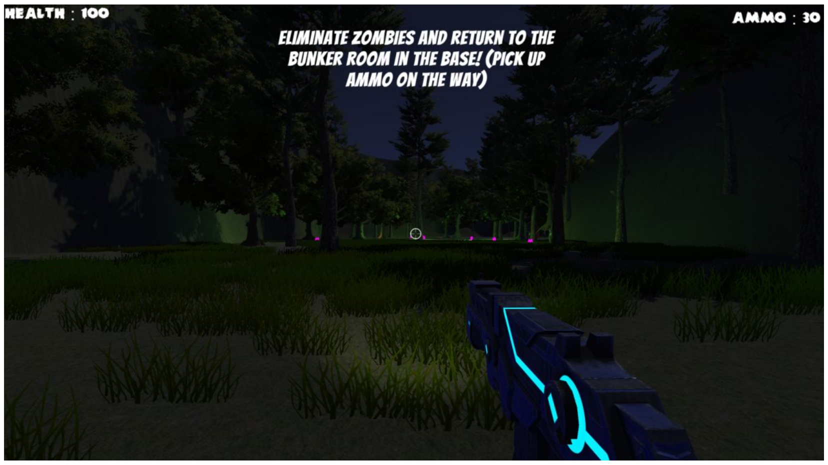
Figure 6: Keyboard Command Use-Case Diagram


Choose gesture settings in Unity
Allows
User
Click to play with settings
Save settings and open Python
Start
Use Hand Force Controller
Simulate keyboard commands

First-Person Shooter Game Subsystem Design

In the early stages of design and implementation, it was suggested to the team by the professor that an open source video game would be an interesting and effective way to demonstrate the capabilities of the Hand Force Controller. A team member was currently working on an online tutorial course for game development in Unity that could be used as an example application (Tristem and Davidson). A video game developed within the team was beneficial to the project's time scope because the source code was more familiar compared to an unfamiliar open source project. The zombie first person shooter (FPS) game was developed in this tutorial course and used free art assets provided with the course and from the Unity asset store. The integration of the Hand Force Controller into this demo game is an example of the Hand Force Controller being implemented directly into a video game by the developer.

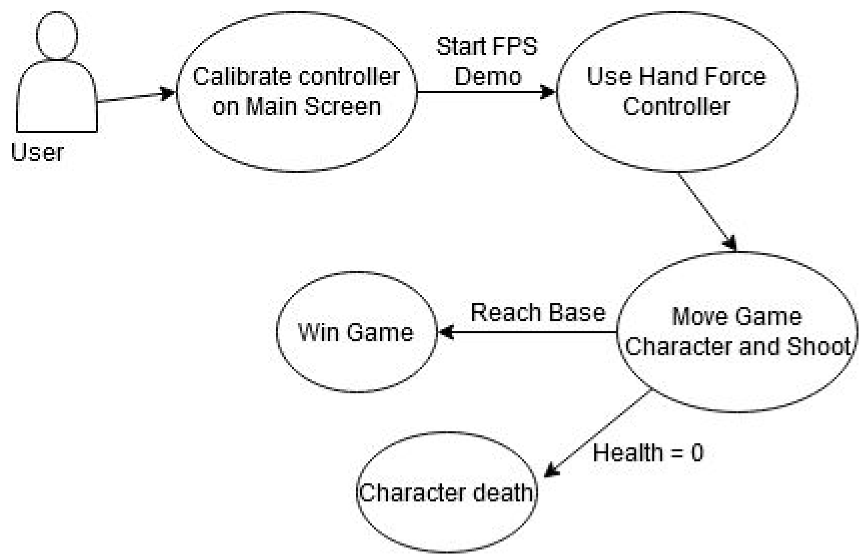
Image 5: First-Person Shooter View


HEALTH : 100
AMMO : 30
ELIMINATE ZOMBIES AND RETURN TO THE BUNKER ROOM IN THE BASE! (PICK UP AMMO ON THE WAY)

The HFController class is used in the FPS game within a configuration script to map game actions to the Hand Force Controller. Implementing the HFController this way allows for the programmer to abstract the HFController logic from the rest of the game. For example, in the HFConfig script the “moveForward” action may be mapped to sensor 2, and therefore, another script that contains the logic to move forward only has to reference the HFConfig.moveForward field and not the actual sensor ID. Controller calibration is implemented in the FPS demo such that the calibration values are retrieved from the Unity PlayerPrefs object and are used as a resting threshold for the sensors.

(Kveton)

Figure 7: First-Person Shooter Game Use-Case Diagram


User
Calibrate controller on Main Screen
Start FPS Demo
Use Hand Force Controller
Move Game Character and Shoot
Reach Base
Win Game
Health = 0
Character death

Testing Results

The serial communication subsystem is the most tested subsystem in the project because every other subsystem is dependent on the communication functioning properly. The testing of this subsystem was performed using a unit testing approach to verify functionality of the various components within the subsystem. As each component of the subsystem was implemented it was tested to meet the necessary functionality in various use cases. This approach allowed for the subsystem to be implemented in a modular way such that refactoring code would have a lower probability of introducing complicated bugs.

Once the basic functionality of the serial communication subsystem was implemented, the subsystem was tested against various use cases to determine what exceptions should be handled to avoid software failure and what programming logic would need to be redesigned.

Early testing and refactoring allowed for this subsystem to be integrated with other subsystems more efficiently.

(Kveton)

Table 1: Serial Communication Testing Outline

Use cases and scenarios	**Results**
Sensor data packets being transferred.	Possibility of raising IndexOutOfRange exception when parsing message in C#. Solution involved clearing the input buffer in case of leftover data.
Serial port name is hardcoded and then a different machine/operating system is used.	Port is not found based on the given name in code giving errors. Solution involved designing an algorithm to programmatically open the port.
Serial data being used to move Unity gameObjects.	Framerate when moving an object becomes very slow and choppy. Core issue is due to the SerialPort.ReadLine() function blocking until data is available to read. Solution involved checking for data before calling the function.
Port finding algorithm is implemented, but without proper port closing logic.	Testing this led to the knowledge that there are two cases in which a port will not connect: 1. A response message was received over the port, but it is incorrect. 2. No response received.
The user quickly stops and starts the Unity application.	The port will not reconnect. This was due to the Arduino code already receiving the “wakeup”. Solution involved implementing a timeout mechanism in the Arduino to return to “sleep” after a period of disconnection.
The user disconnects and reconnects the controller during Unity application runtime.	IOException thrown causing crash. Solution involved implementing an IsConnected variable and a ReconnectDevice() function in

	HFController.
Game logic is programmed to use the HFController.GetSensorValue() function when the port is not open.	IOException thrown causing crash. Solution involved wrapping HFController code logic in an if statement using HFController.IsConnected.

Discussion on Lessons Learned

Lillian Boettcher:

During this project I learned a great deal about hardware to software connections. As a programmer I had little hardware experience in terms of consolidating hardware and ensuring it works as effectively as a software. And while I had used Unity before to develop, it is my first time doing it as a team, which will probably serve me well in future programming projects throughout my career that rely heavily on collaboration.

Grant Kveton:

Working on the Hand Force Controller project has given me a good experience of designing, implementing, and testing a software application from start to finish. Before the Engineering Design course, I had some basic experience and interest in using the Unity game engine. During the research and design stages of the project, I learned a lot about the Arduino documentation, C# documentation, and standards for the Unity game engine. I had an interest in testing practices and wanted to integrate that into the components of the project I was working on. Overall, I learned a lot about the importance of writing clean modular and testable code and the effects on debugging and development. I also gained experience in working with a team on a large software project such as this. Given the state of affairs with COVID-19 I was exposed to working with the team remotely and the importance of communication.

Dhaval Manojkumar Shirvi:

While working on this project, I learned a lot about Unity as it was my first time developing in unity. While working on the keyboard emulator, I picked up on a lot of the unity's UI mechanics. I have used python before, however it was my first time working with ironpython. Due to COVID-19 and me being in India, I learned a great deal about time management due to different time zones, and the importance of communication as we were working on the project remotely. It was a great experience to work on this project with my team.

Conclusion

The project ended successfully, meeting all of the initial functional and nonfunctional requirements set. The project design changed slightly throughout, but only a relatively small amount and all changes made were to better follow the requirements and goals of the project.

Feedback on the project is fairly positive. The development team finds the Hand Force Controller easy to use and other individuals found it also easy to use. One criticism by a non-developer tester was that having a usable platform required having access to a 3D printer when the goal of this project is to have the controller as accessible to the world as possible. One way that this could be addressed is by including a DIY (do-it-yourself) instruction manual for assembling a platform out of construction parts, such as cardboard. This could also turn assembly into a fun project and encourage developers and users alike to become more involved in the hardware assembly even if they do not put the hardware together themselves.

Additionally, the project could be improved with the use of standardizing components. The jump wires used to go from the Arduino Uno to the breadboard circuits are very large and bulky, making the hardware hard to consolidate with a platform. Additionally, finding force sensors that can be responsive to fingers may be difficult for some users who wish to build the Hand Force Controller. Therefore, people may benefit if the repository also includes a list of recommended parts for the user to use, which would not just consist of the parts the developers used but also a list of parts with minimal cost and maximum effectiveness for the purposes of the controller.

(Boettcher)

References

Bierre, Kevin, et al. “Game Not Over: Accessibility Issues in Video Games.” *International Game*

Developers Association, 2005.

Cook, David, et al. “Virtual Reality and Older Hands: Dexterity and Accessibility in Hand-Held VR Control.” *Association for Computing Machinery*, vol. 5, 2019, pp. 147-151.

“Force Sensitive Resistor (FSR).” *Adafruit*, 2020, learn.adafruit.com/force-sensitive-resistor-fsr/overview.

Glinert, Eitan. “The Human Controller: Usability and Accessibility in Video Game Interfaces.” *Massachusetts Institute of Technology*, 2008.

“Hardware and Software Specifications.” Unreal Engine, *Epic Games*, 2020, docs.unrealengine.com/en-US/GettingStarted/RecommendedSpecifications/index.html.

Iacopetti, Fabrizio, et al., “Game Console Controller Interface for People with Disability.” *International Conference on Complex, Intelligent and Software Intensive Systems*, 2008, pp. 757-762.

“Keyboard 0.13.5.” Python, *Python Software Foundation*, 2020, pypi.org/project/keyboard.

Kry, Paul, et al., “HandNavigator: Hands-on Interaction for Desktop Virtual Reality.” *Virtual Reality Software And Technology*, vol. 8, 2008, pp. 53-60.

“Language Reference.” *Arduino*, 2020, www.arduino.cc/reference/en.

Liechti, Chris. “pySerial API.” *pySerial*, 2017, pyserial.readthedocs.io/en/latest/pyserial_api.html.

Maggiorini, Dario, et al., “Evolution of Game Controllers: Toward the Support of Gamers with Physical Disabilities.” *Computer-Human Interaction Research and Applications*, Springer Nature, 2019, pp. 66-89.

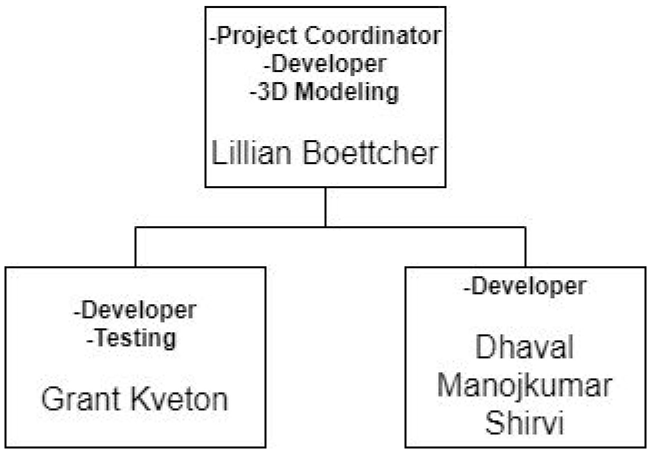
“SerialPort Class.” *Microsoft*, 2020, docs.microsoft.com/en-us/dotnet/api/system.io.ports.serialport.

Tristem, Ben, and Rick Davidson. “Complete C# Unity Game Developer 3D.” *Udemy*, 2020, www.udemy.com/course/unitycourse2.

“74HC4051 Product Data Sheet.” *NXP Semiconductors*, 2016, cdn.sparkfun.com/assets/learn_tutorials/5/5/3/74HC_HCT4051.pdf.

Organization Chart

Figure 8: Organization Chart



Budget

The Hand Force Controller is a cost-effective project due to its use of minimal hardware components and free-to-use computer software. The original part list only included five force sensors and did not include a multiplexer. During the implementation stage, the constraint of only having six analog inputs on the Arduino microcontroller was handled by introducing a multiplexer to include more sensors in the system. The main software programs used in development, such as Unity and the Arduino IDE, are free-to-use in the context of educational projects.

Due to the current state of affairs with COVID-19, the team was not able to work on the project together on campus. Extra parts were ordered so that each team member could work on their own individual hardware setup. It was important for the team members to have identical parts and configurations to reduce the potential for error. Table # below shows the component and total cost for a single Hand Force Controller hardware setup.

(Kveton)

Table 2: Project Budget

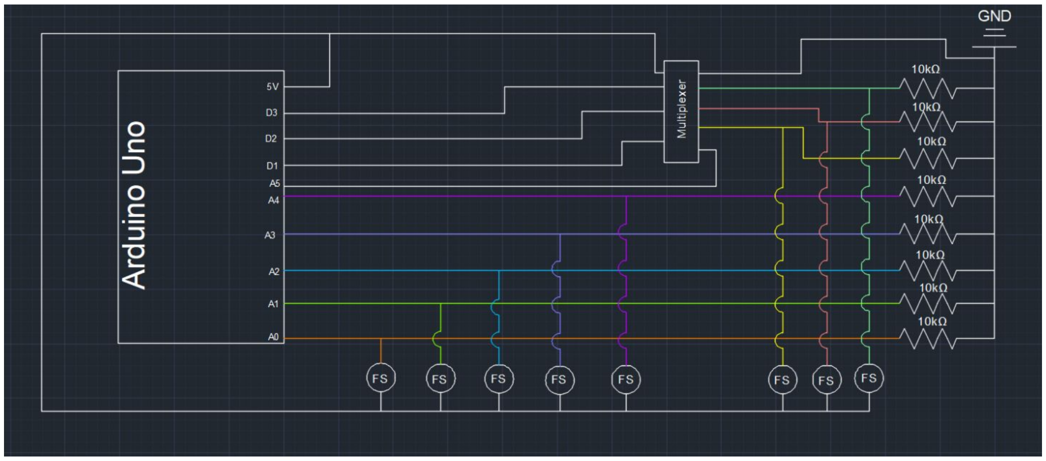
Part Number	Description	Quantity	Unit Price	Total Cost
ARDUINO UNO R3	Arduino Uno Microcontroller Board	1	$22.00	$22.00
FSR400	Force Sensitive Resistor	8	$6.60	$52.80
74HC4051	Multiplexer Breakout	1	$5.95	$5.95

-	-	-	-	**$80.75**

Appendices

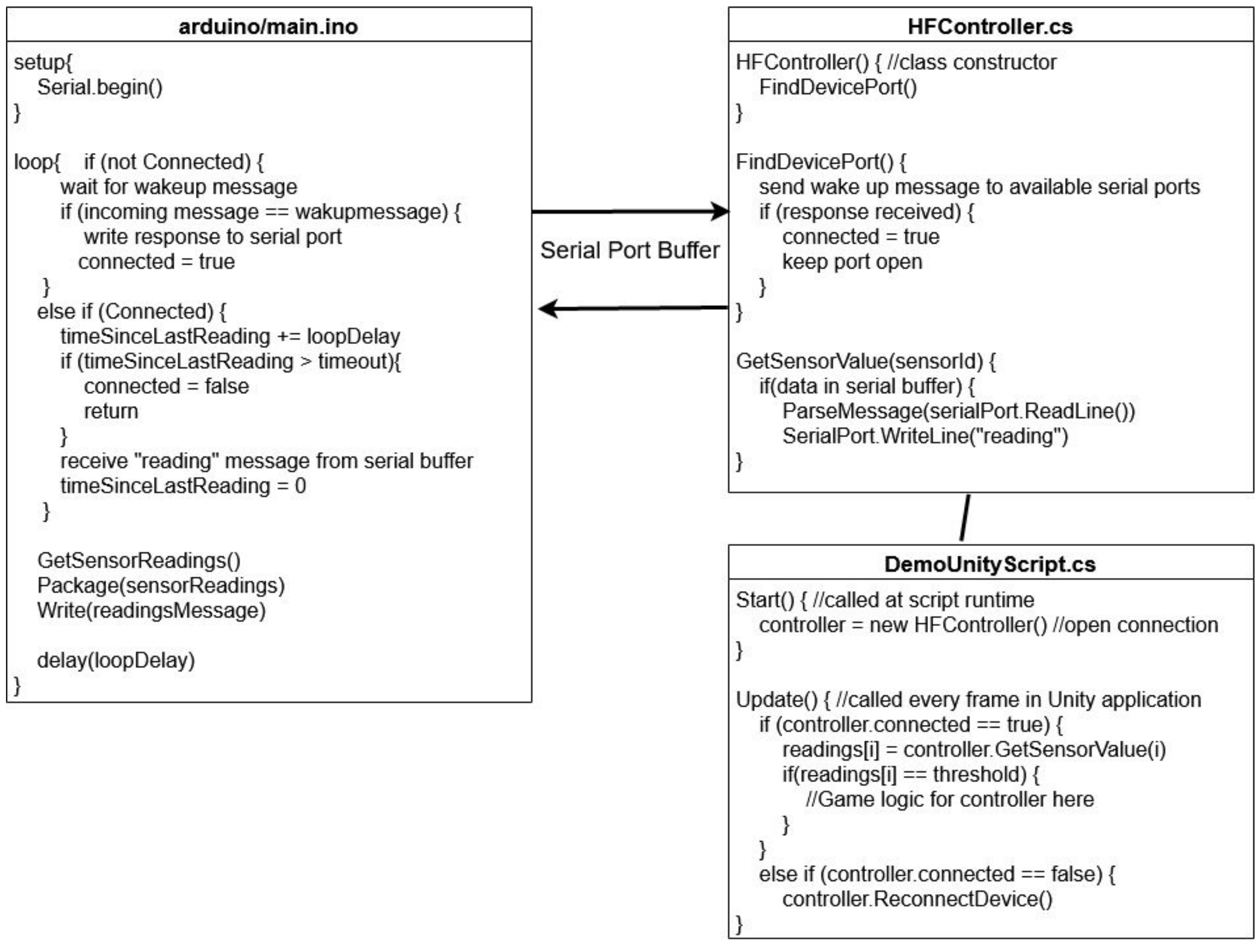
Hardware Schematics

Figure 9: Circuit Diagram



Program Schematics

Figure 10: Serial Communication Pseudocode Schematic



arduino/main.ino

```
setup{
    Serial.begin()
}

loop{    if (not Connected) {
        wait for wakeup message
        if (incoming message == wakupmessage) {
            write response to serial port
           connected = true
     }
    else if (Connected) {
        timeSinceLastReading += loopDelay
        if (timeSinceLastReading > timeout){
            connected = false
            return
        }
        receive "reading" message from serial buffer
        timeSinceLastReading = 0
     }

    GetSensorReadings()
    Package(sensorReadings)
    Write(readingsMessage)

    delay(loopDelay)
}
```

HFController.cs

```
HFController() { //class constructor
    FindDevicePort()
}

FindDevicePort() {
    send wake up message to available serial ports
    if (response received) {
        connected = true
        keep port open
    }
}

GetSensorValue(sensorId) {
    if(data in serial buffer) {
        ParseMessage(serialPort.ReadLine())
        SerialPort.WriteLine("reading")
}
```

DemoUnityScript.cs

```
Start() { //called at script runtime
    controller = new HFController() //open connection
}

Update() { //called every frame in Unity application
    if (controller.connected == true) {
        readings[i] = controller.GetSensorValue(i)
        if(readings[i] == threshold) {
            //Game logic for controller here
        }
    }
    else if (controller.connected == false) {
        controller.ReconnectDevice()
}
```

Figure 11: Application Main Screen UI Design

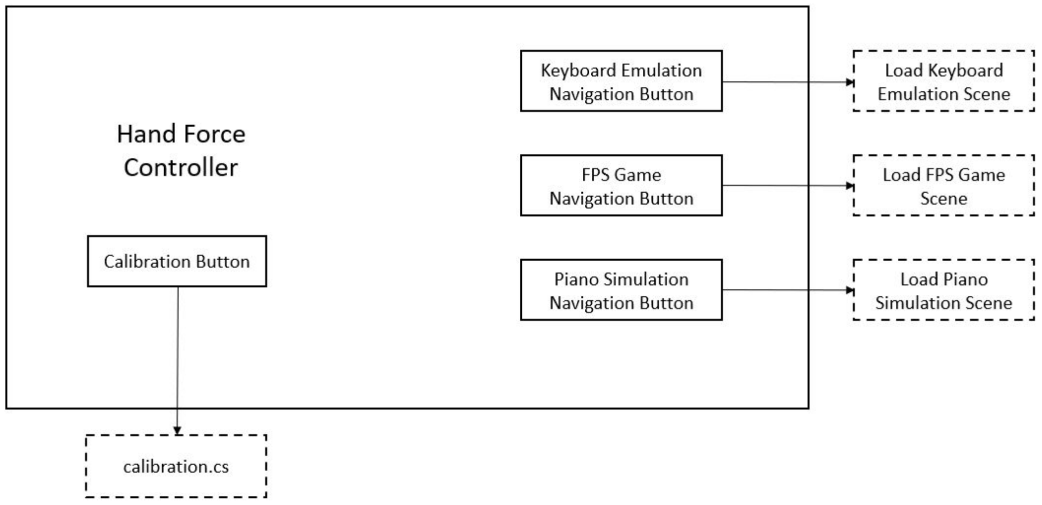

Hand Force Controller
Calibration Button
calibration.cs
Keyboard Emulation Navigation Button
FPS Game Navigation Button
Piano Simulation Navigation Button
Load Keyboard Emulation Scene
Load FPS Game Scene
Load Piano Simulation Scene

Figure 12: Calibration Activity Diagram

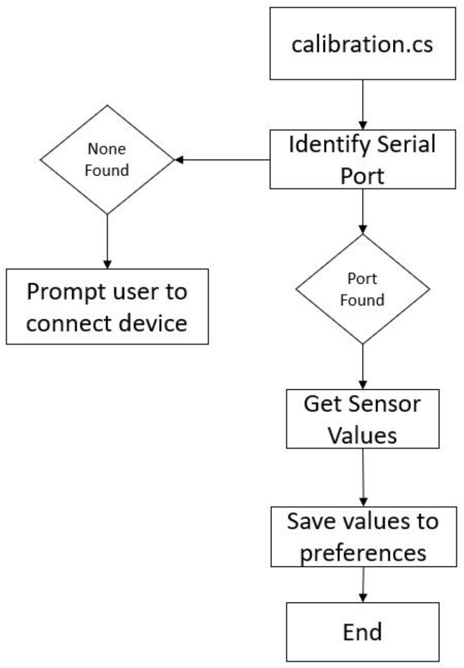


Figure 13: Keyboard Emulation Activity Diagram

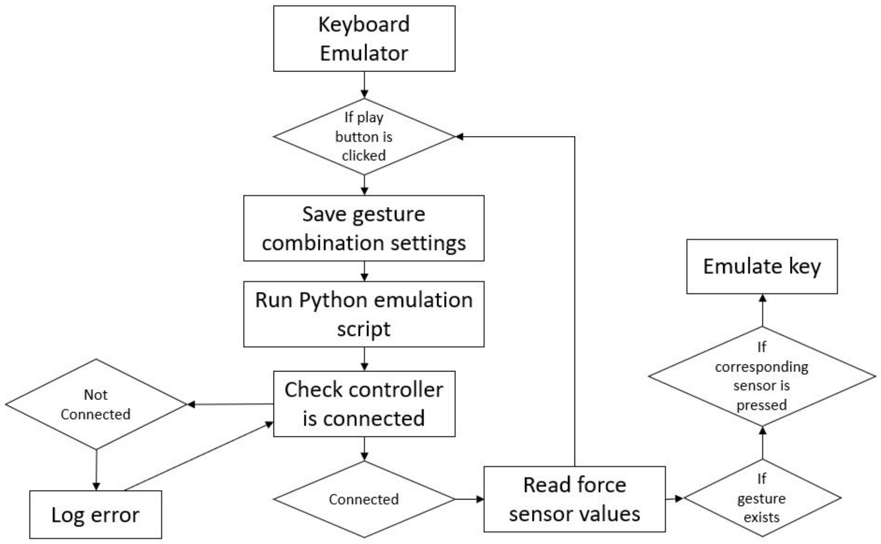

Keyboard Emulator
If play button is clicked
Save gesture combination settings
Run Python emulation script
Check controller is connected
Not Connected
Log error
Connected
Read force sensor values
If gesture exists
If corresponding sensor is pressed
Emulate key

Figure 14: Piano Simulation Activity Diagram

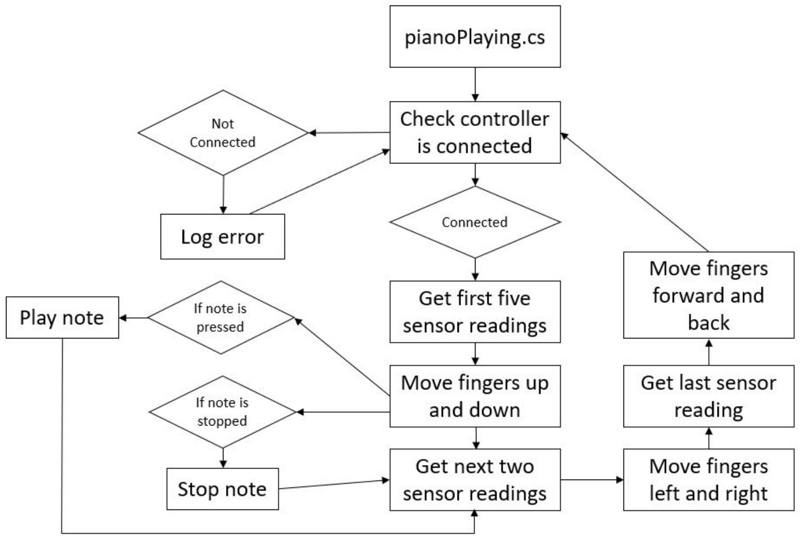
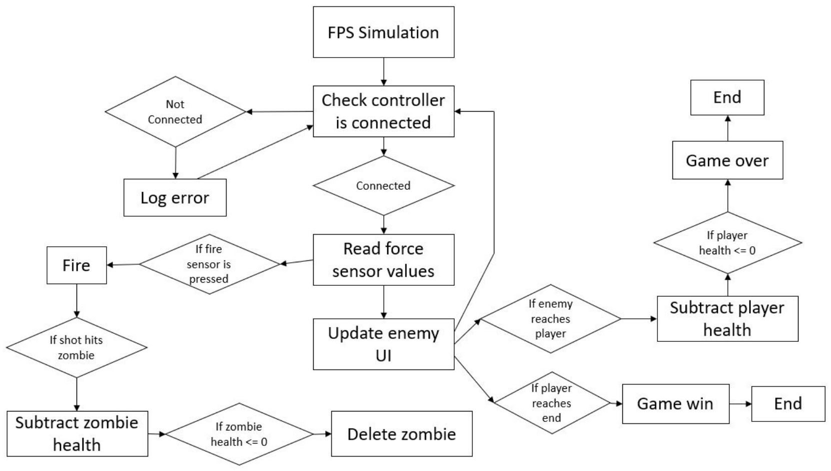


Figure 15: First-Person Shooter Game Activity Diagram


FPS Simulation
Check controller is connected
Not Connected
Log error
Connected
Read force sensor values
If fire sensor is pressed
Fire
If shot hits zombie
Subtract zombie health
If zombie health <= 0
Delete zombie
Update enemy UI
If enemy reaches player
Subtract player health
If player health <= 0
Game over
End
If player reaches end
Game win
End