Final Interface Specification
VR Tightrope Simulation

Summary: Our Tightrope Simulator consists of two separate hardware devices that require constant interaction to determine a positive user experience. As such, the Tightrope Simulator can effectively be divided into two equal components: the Arduino and the Oculus Rift. The specifications for how these devices interact are documented below.

Architecture: This project multiple hardware components: an Oculus Rift virtual reality headset, an Arduino embedded system, and PWM-based pressure sensors. A Windows PC serves as a medium between the two to communicate data. The physical configuration can be seen in the below diagram.

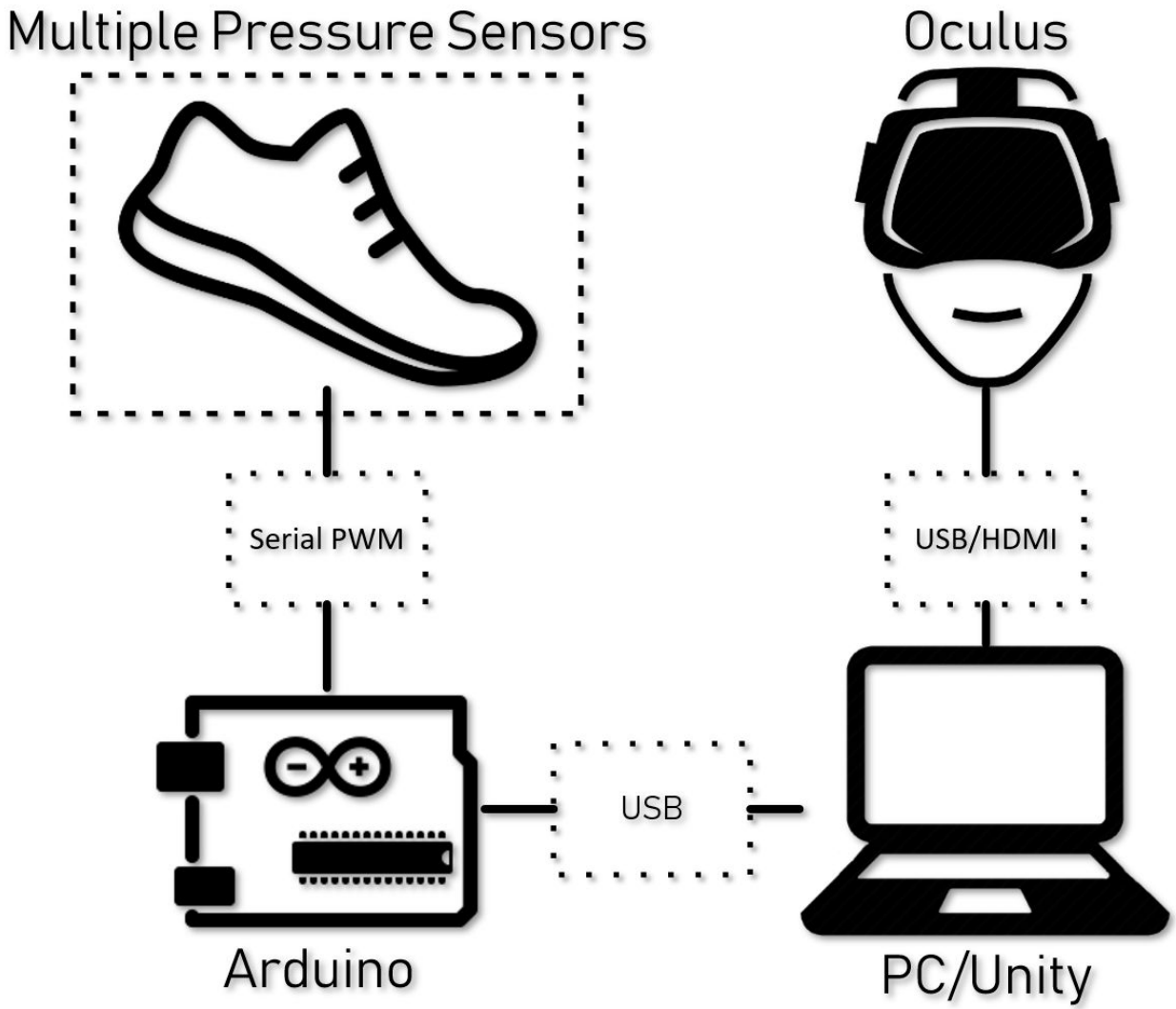

Diagram 1: Non-Technical Configuration

The Unity Engine runs in the background on the Windows computer, and runs all of the code for this project. Written in C#, it contains the ability to render 3D data to the Oculus headset, and read data from the Arduino.

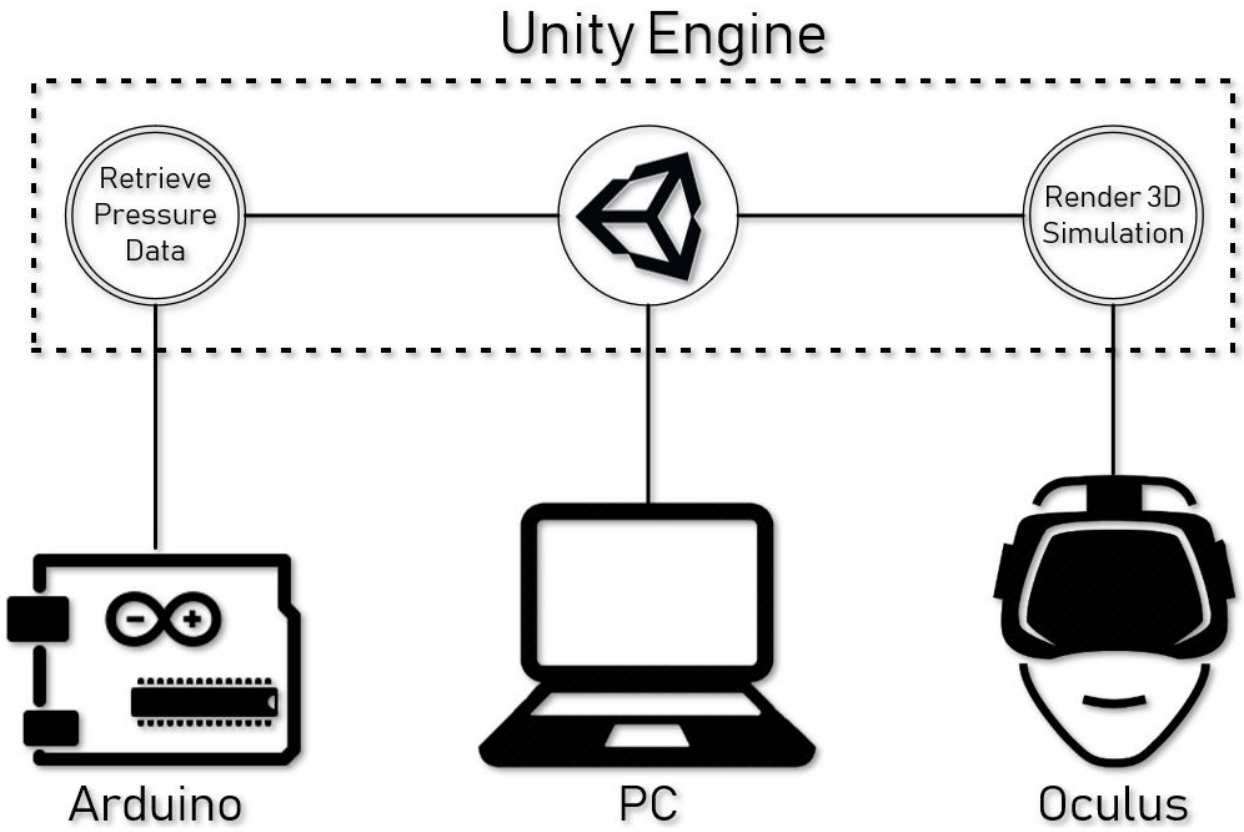

Diagram 2: The Unity Engine Hardware Setup

The Oculus Rift headset is connected to the Windows computer via an HDMI and USB cable. The Unity Engine can render a 3D digital scene to the headset and immerse the user in this experience via a fully interactive view. The Unity Engine also contains a USB library to read data over a serial connection from the Arduino. An event driven approach will allow data to be polled as the user interacts with the simulation and be used to provide feedback to the users' performance. This polling will be done asynchronously in this application so that the 3D simulation can still be seamless.

When data is polled from the Arduino, a request must be made. This will require locking a mutex on the serial USB connection so that there cannot be any overlap or malformation of data. The Arduino will receive the request for data and respond with an acknowledgement of the request, and then the requested data..

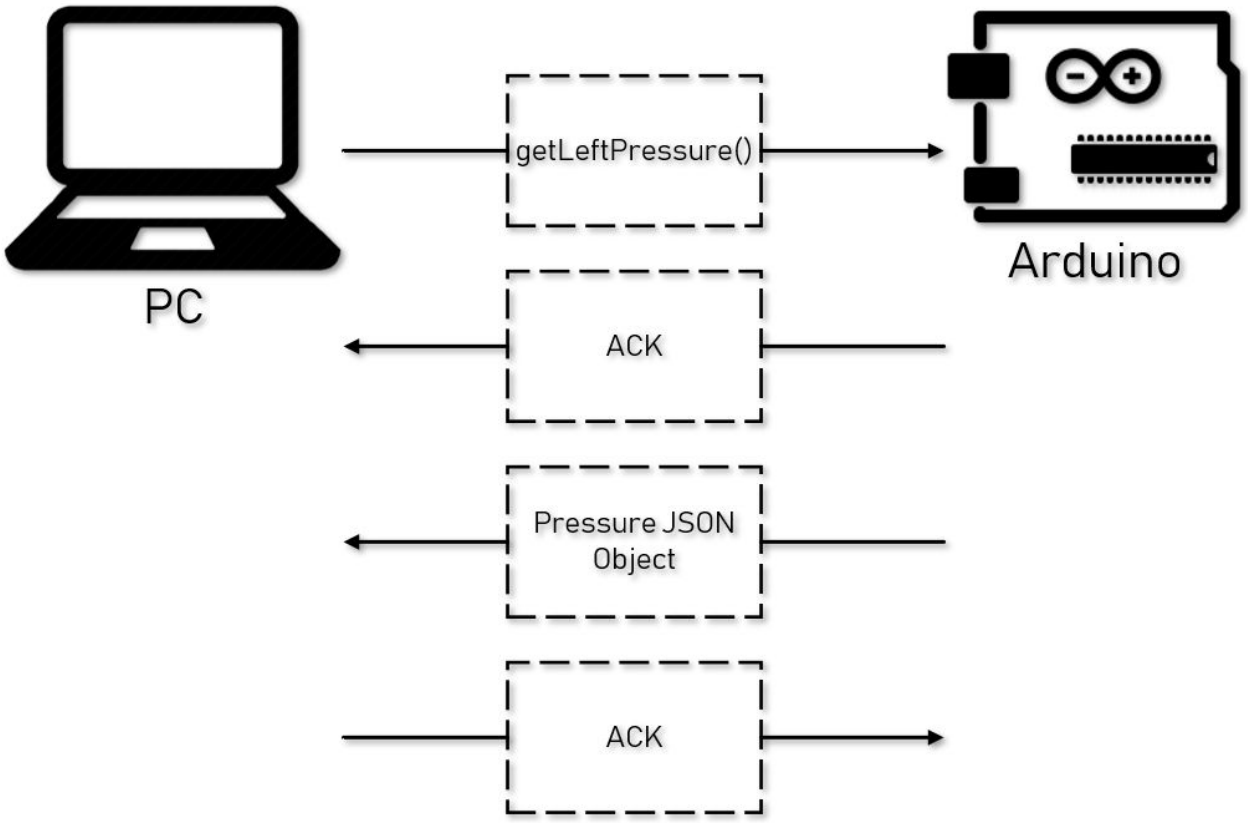

Diagram 3: Data Polling

These two devices utilize a custom reliable-communication protocol that ensures data is delivered properly and is non-malformed.

Data Communication Protocol: In order to ensure data can be properly transmitted between the Unity Engine and the Arduino, a custom protocol must be developed. The framework for this protocol is similar to the reliable-communication TCP protocol that is used in the TCP/IP stack. The packet design for the custom Arduino protocol is as follows:

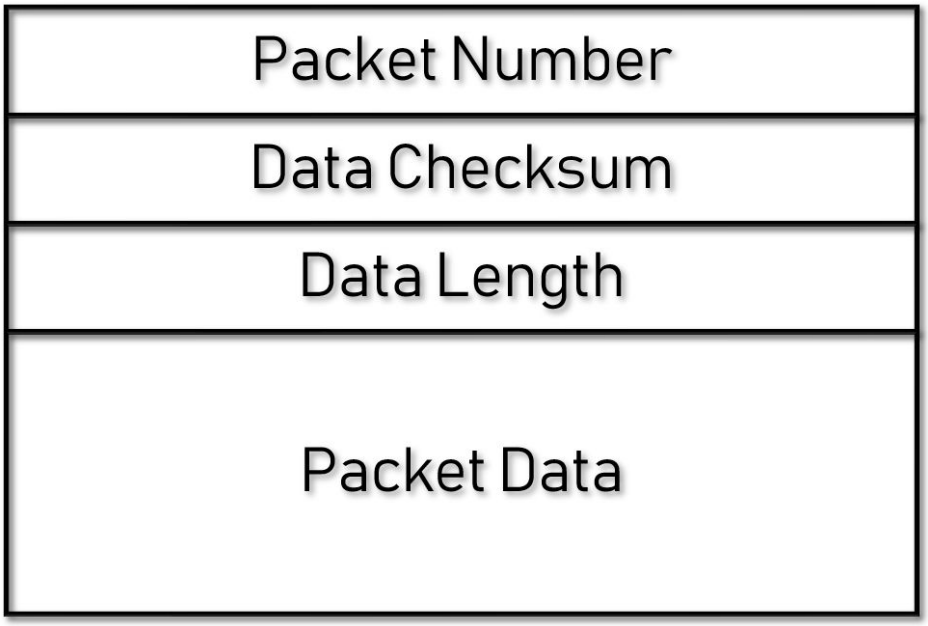

Diagram 4: Data Communication Protocol Packet

If we unpack this protocol, a typical transmission between the Unity Engine and the Arduino may look like the following:

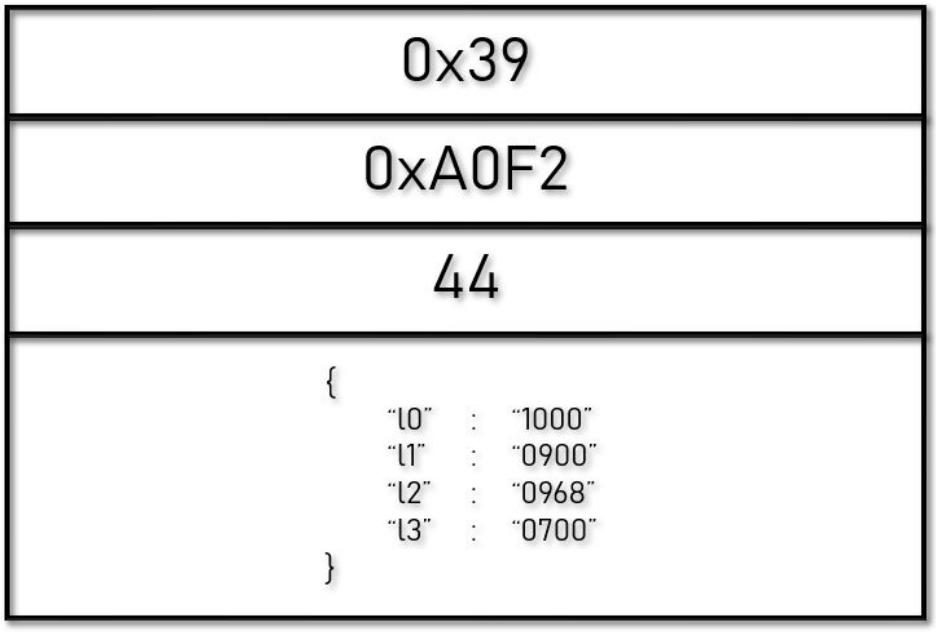

Diagram 5: Packet example

Arduino:

The Arduino will be connected directly to the pressure sensors via a PWM 3-pin wire. This allows it to poll the pressure sensors and receive analog data. This data would likely be a value

between 0 - 1000, and would represent at what percentage of pressure the sensor is under. This data can be calibrated to be able to determine when an individual has their foot in the air, or on the ground.

The Arduino calls a standard library function, analogRead(int PORT), where the port represents the analog port on the Arduino board. This will retrieve the PWM reading of the sensor on this port. This data will be compared to a previous polling, and at a certain delta can be calibrated to determine what the stance is of a users' foot. This will be polled every 100 milliseconds to provide realistic feedback, and will be tweaked in order to improve the user experience. At a certain threshold, this will trigger an alert on the Oculus headset. This information is further explained in depth in the Oculus/Unity section.

While the Arduino is a robust, consistent, sturdy device, it has one major flaw that may hinder the progress of this project: a lack of disk space. Much of the aforementioned processing may need to be done on the Unity side as only a certain amount of code can be compiled to the Arduino. In addition, reliability methods such as packet information processing may have to be simplified in order to speed up data transmission. Further integration between the two parts of this project and testing will reveal the gravity of this situation.

The basic functions and their parameters that the Arduino will be using are documented below:

`String getLeftPressure(void):`

Retrieve the pressure sensor readings from all sensors on the left foot. Transform this data into a JSON object. Put data into custom packet and transmit over serial.

`String getRightPressure(void):`

Repeat of getLeftPressure, except for right sensors.

`int GetArduinoStatus(void):`

This function is called when building the initial communication between the Arduino and PC (Unity Engine). This function runs a check on the Arduino to ensure it is properly communicating data. It will send a test packet to the PC and expect a response test packet. Returns 0 if it cannot communicate with the PC.

`int GetSensorStatus(void):`

This function runs a check on each of the individual sensors on both feet to ensure that they can return properly calibrated data and are functioning as expected. Returns a binary string of 1's or 0's to determine the status of each sensor. This string's length equals the amount of sensors.

`void CalibrateArduino(void):`

This function runs a procedure on the Arduino for calibrating each individual sensor. This data would be communicated through the Oculus headset to the user as a mini-tutorial where they would put pressure on each individual sensor.

`void CmdHandler(String data):`

This function ingests data over the Serial USB line and determines what action to perform. It does this by unpacking the packet, validating its integrity, and then executing the appropriate command. Before executing the command, it sends an acknowledgement to the PC via the serial data line.

Oculus, Unity:

Rendering Environments on the Oculus Headset:

The application will be seen through the Oculus headset due to the Oculus API that is available for use in the C# scripts in Unity. There will be a first person controller object put into the scene that will hold a script that makes it so that the camera follows the first person view of that player gameobject. This way the scene, with the virtual reality option turned on in the settings of the Unity scene, would feel like they are actually in the scene. The headset will be able to visualize the 3D environment of the scene from the player object's point of view.

Receiving the Arduino Data:

Unity has a class called SerialPort that will be used to communicate with the Arduino. The data will be retrieved via a C# script that runs a communication protocol that will be able to constantly ask for data in real time. When the Arduino receives a signal from the Unity C# script, it will respond by sending back data from the foot pressure sensors. This data will be in the form of integers that will correspond to each of the sensors on both of the feet. The function that receives this data will return that data as JSON object, that will be dissected when ready to be processed. More specifically, the values from each pressure sensor will be extracted into their own integer variables in order to be processed in the next stage of the script's cycle. This script will be constantly running and checking to receive data every 100 milliseconds.

Processing the Arduino Data (The Feedback Loop):

In order to make sure the user has the correct footing on the virtual tightrope, the Unity C# script will have a series of conditions that the pressure data will have to go through that will depend on its value and the position of that specific sensor on the foot. The condition that the values satisfy will then elicit a response to the user in the form of text. The script will be attached to a graphical user interface component (such as a speech bubble looking text field) that will appear on the side of the screen. This advice outputted in turn will reflect what the user is doing wrong or right in this case depending on the condition the values from all of the sensors satisfied. The graphic user interface component would be able to display the advice text through Unity's graphical user

interface API. The user will be able to use the Oculus to look around them using head tracking technology while the user interface continues to follow in order to stay in their field of vision. This entire concept is what is termed as the feedback loop for this application.

Determining the Frequency of Feedback Output:

To ensure the application isn't constantly outputting advice (text messages) to the graphical user interface component, the Unity application will only output feedback to the user if there are large discrepancies between data sets in order to respond quickly. These data sets will be judged based on certain thresholds between current and previous sets. If these thresholds are overtaken by the numerical differences between the data sets, then the application will be prompted to output a new response into the feedback loop. In a more simple sense, this way the simulator will understand that the user is veering off course and respond appropriately. This would allow for less repetitive and common feedback responses when considering only slight changes in the pressure data sets. The only time the advice will change is if it is necessary, which will contribute to making the user experience less distracting and confusing. In summary, the pressure data will be received from the Arduino as the script is constantly running on the graphical user interface component the application, the data will be checked to make sure it is significantly different than the values previously received (from the last data set), and if so, that data will be put through many varying conditions until it matches the correct advice that would correspond to the user's current footing position. Below is a concept image of what this type of interface will look like with the feedback loop enabled.

Diagram 6: User Interface Mock-Up