Human Motion Sensing Unit User Manual

Iteration 2

XProjects Spring 2021

Minna Kuriakose, Victoria Turchick, Ethan Vukelich, Amelia Wear

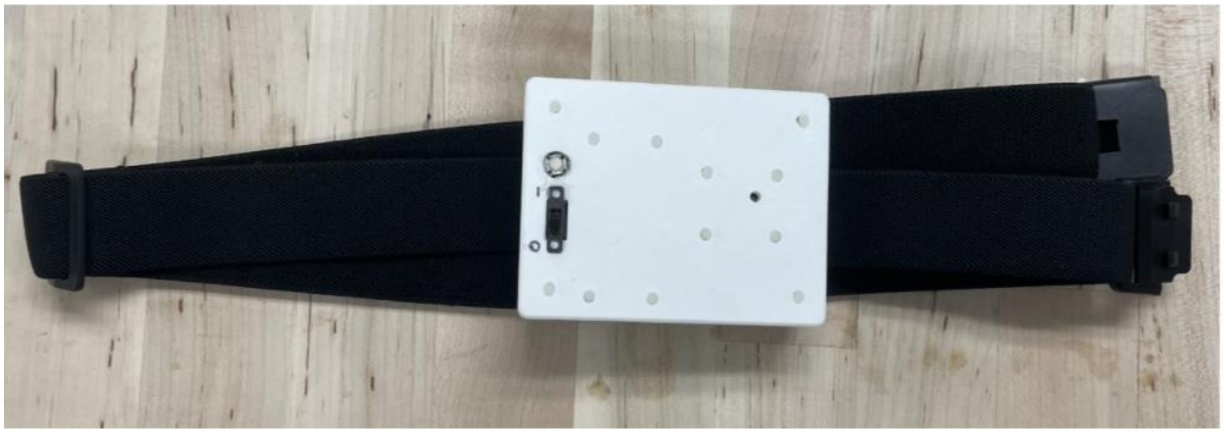
Table of Contents

I. PURPOSE

This document serves as a continuation of the Human Motion Sensing Unit (HMSU) XProject from the Fall of 2020. It will overview the purpose, assembly, and use of the updated product produced by the HMSU2 XProject team.

The HMSU is a wearable device that measures, and outputs select IMU data in real time, including acceleration values and orientation quaternions. The purpose of this device is to evaluate human movement in real time for applications in repeatable subject testing and rehabilitation. Currently, data will be transferred via serial port, but this document will discuss and share preliminary code on transferring data via Wi-Fi to allow use of the device in a less invasive manner. The device will sit on the lower back of the user, be battery-powered, and self-calibrating.

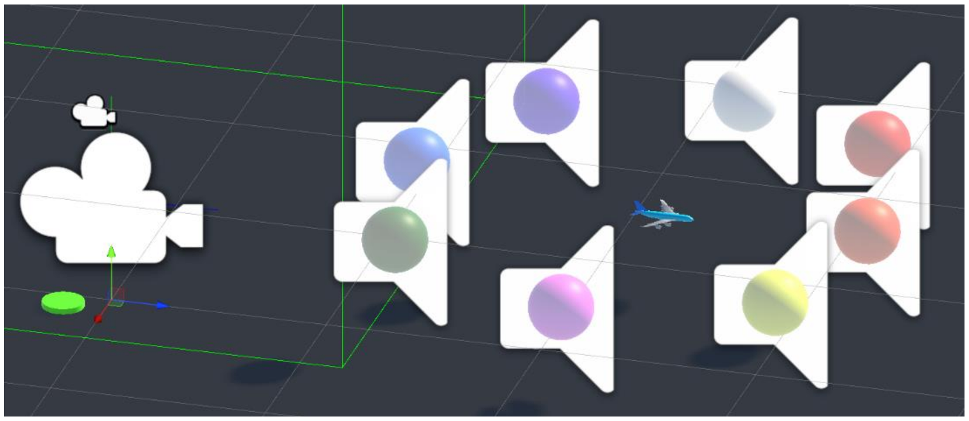
The team has also developed a 3D virtual reality environment to accompany the HMSU that researchers can use to conduct human movement experiments. This Virtual Reality (VR) environment will also output the data from the device so that researchers can use the record for further analysis. The intent of the design of the HMSU and the VR environment is to be open source and easily replicable.



II. KEY FEATURES

The HMSU device is comprised of an Adafruit Huzzah and an Adafruit BNO085 IMU. The device sends data serially via a USB cable to a Unity virtual reality application for testing purposes. The device sends acceleration and orientation quaternions to the VR application which are also written to a text file to be accessed by the researcher later and used to determine orientation and basic positioning for the user within the VR application.

The VR application allows the user wearing the device to walk and reach pre-set, sphere-shaped targets. The user will be able to see themselves within the application with the help of an offset airplane-shaped cursor located in front of the device. Upon the cursor making contact with a target, the user will hear a sound via the VR headset and the target will turn green in color signaling a successful contact.



The data is sent serially via a micro-USB cable and the data transmission rate is approximately 100 Hz. The device is rechargeable by plugging in the micro-USB cable. The gyroscope, accelerometer, and magnetometer are all calibrated internally by the BNO085.

III. USE CASE OVERVIEW

The use case describes the expectations of a single use of the system. It outlines the process of using the system from the time a subject steps into the laboratory to the time they leave.

Step 1: Subject steps into lab. Researcher hands the subject a belt and asks them to attach it to their midline, under their shirt, and adjust it so the belt is snug under their clothing; the gripping fabric should be comfortable against their back to prevent movement of the belt during the trial. The subject moves to the starting point of the test and dons a VR headset.

Step 2: Researcher turns on sensor, lays it flat on a table, leaves the device motionless for five seconds (first initialization), and then attaches the device to the Velcro on the belt. Indicator lights on the device show it is turned on and operating normally.

Step 3: The subject should wait another 10 seconds for the second initialization period to pass. The VR application is now fully operational, and the researcher can then instruct the patient through the steps of the exercise.

Step 4: The researcher views the game via the screen of the host computer.

Step 5: The experiment concludes, and the researcher stops the game. The subject then takes off any VR equipment.

Step 6: The researcher can remove the sensor from the patient and turn it off. The subject can remove the belt from their torso and leave.

Step 7: Researcher can view IMU data via an outputted text file on their computer.

IV. BILL OF MATERIALS

	Item	Quantity	Price (USD)
Housing and Patient Attachment	Adjustable Elastic Belt	1	5.66/belt
	Velcro	1	8.99
	Grip Fabric	1	12.00
Hardware	Nuts #2-56 (Nylon),	2*	7.14
	Screws: #2-56 (Nylon) (1”)	2*	6.69
	Slide Switch	1	0.58
	Threaded Inserts	4**	13.99
	Power Converter	1	4.95
	3.7v 1200mAh Battery	1	9.95
	Push Button	1	Variable models
Device	Adafruit Huzzah	1	9.95
	Adafruit BNO085 IMU	1	19.95
Total Cost			$99.85

*Came in a pack of 100, **Came in a pack of 50

V. OPERATIONAL PROCEDURES

This section outlines the standard operating procedure of the system once all setup is complete (outlined in sections VI and VII).

Attach the device to the user under their shirt around the lower back. Make sure it is nice and snug (adjust the belt as needed) and that the device is sitting flat and upright against their back. If data is being transferred via serial port, ensure that the micro-USB is long enough to allow ample movement. This cable can also be run alongside any cables for the VR headset to ensure it does not hinder movement.

Make sure that both initialization periods have passed before starting to collect data. During the first initialization period, the device should lay still on a flat surface. This can be done in any orientation or position, as long as the device is sitting still. Attach the device to the user only after the first initialization period has passed. Wait for the second initialization period to pass before starting the experiment.

If device does not connect properly to the Arduino serial port monitor, try a combination of pressing the reset button and power-cycling the device; this restarts the serial communication. This error will show up in Unity as either a COM Port not available or a TimeOut error. Similarly, if the Arduino Serial Port Monitor prints that it cannot initialize the BNO085, power cycle the device and press the reset button.

When you would like to turn off the system, simply turn off the switch on the device and stop the VR application on your computer. The orientation and positioning data from the session will be written to a file called "OutputData.txt" in the Documents folder of the computer. This will be cleared and overwritten every time the VR application is played.

VI. SETTING UP THE SOFTWARE

Please refer to this section for initial software setup of the device. All necessary code developed by the team can be found in the HMSU GitHub here: https://github.com/aw58/HMSU2.git

i Software Dependency Preparation

Complete this entire list and either ensure each item is installed, or follow the instructions provided at the website.

- (Windows 10 or higher)
- A connection to a private Wi-Fi network.
- Arduino IDE for Windows: https://www.arduino.cc/en/software
- Arduino Libraries and Packages
  - First, follow the instructions on this page: https://learn.adafruit.com/adafruit-feather-huzzah-esp8266/using-arduino-ide
  - Install the Adafruit_BNO08x library, accessible in the Library Manager window
- Unity 2019.4.20f1 (for editing purposes)
  - The free download of Unity is sufficient to run, change, and update the VR application.
  - Using other specific versions of Unity (such as 2019.3) will cause errors in compilation. If you would like to use a different version, you will most likely have to create a new project for the version of your choice. In this case, follow the instructions in the Unity section of this paper.
- If you are making the VR environment from scratch or would like to change any code, download all files in "FinalDownloadables" from the HMSU GitHub repository: https://github.com/aw58/HMSU2.git

VII. FABRICATION

i. Electronics Hardware Preparation

First, connect the Huzzah to the IMU via I2C connection with the following steps. Wiring may be done with either hardwires soldered directly to the BNO085, or with a STEMMA QT connector, the recommended way to route wires to the IMU. Wire the IMU's VIN wire to the Huzzah 3V pin, and GND to GND. Next, connect IMU SCL wire to the Huzzah SCL and IMU SDA to the Huzzah SDA. Diagrams and more directions for this I2C wiring can be found here.

ii. Implementing Code

Once the Arduino IDE is setup according to the Software Dependency Preparation section (VI. i), the code may be uploaded to the Huzzah. First, be sure to choose the correct board from the board manager list. The Huzzah's board settings should be set to run at an upload speed of 115200, and a CPU Frequency of 80MHz; all other settings can remain as the defaults.

Before uploading the code to run, be sure that your Serial Port is set to run at a 15200 Baud rate. Change this by plugging in the Huzzah, opening the serial port in the upper right corner of the IDE, and selecting 15200 from the drop-down menu in the bottom right corner of the screen that appears.

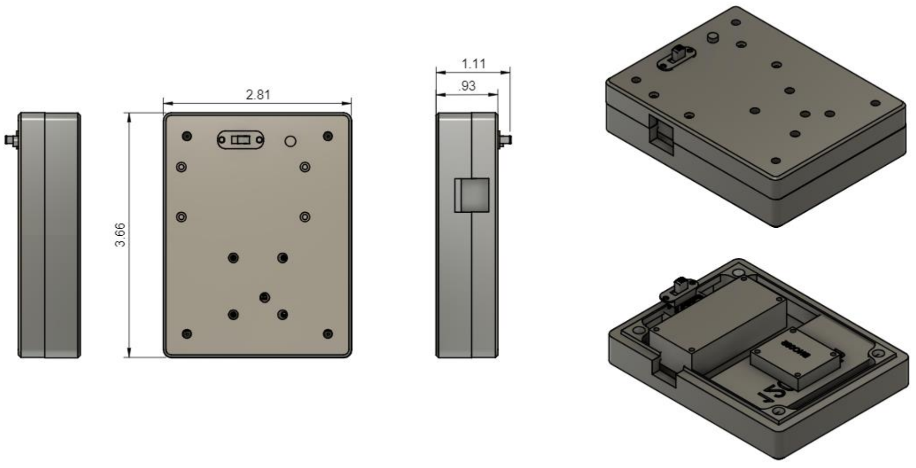
To use the project as created by the HMSU2 team, you may download the .zip file here and follow the instructions here to add it to Unity Hub.

iii. 3D Printing

Necessary CAD models can be found in the GitHub Repository under the "CAD Models" folder. In order to make these models as accessible as possible, they are in found in Fusion 360 format. Assembly parameters have been used to define most dimensions within the models, and may be altered according to individual needs, which will adjust the rest of the model accordingly. Each device requires two prints: the top and bottom to the case. These prints are recommended to be executed with a 20% infill.

iv. Full Assembly

The purpose of this section is to outline the assembly of the device, after the completion of the steps included in the Electronics Hardware Preparation and 3D Printing sections.



1. **Threaded inserts**

Four threaded inserts should be put into each of the four holes on the inside of the case bottom. These inserts should be fixed to the case using a soldering iron: touch and hold the iron to the inside of the insert long enough to melt the surrounding plastic and adhere the inserts to the case. The inserts will sink down into the holes but should not be pushed so far as to have the upper portion of the insert covered by the melted material above it.

2. **Adding the switch and reset button**

There are three lugs on the bottom of the switch. Wire leads to two consecutive lugs (middle and right or left). At this point, the switch can be attached to the case at the switch cutout with hot glue. Solder one of the wires to the ground (GND) pin, and the other to the enable (EN) pin on the Huzzah.

To attach the reset button, wire leads to the bottom of the button, and attach the switch to the case via hot glue. Then wire one lead to GND and the other to reset (RST) on the Huzzah.

3. **Mounting the Huzzah and the IMU**

Once the IMU and Huzzah are connected according to the hardware preparation section (VII. i) , they can be attached to the case. The Huzzah should be inserted into the top of the case, oriented such that the USB port aligns with the USB cutout of the case. The board can now be fixed to the case using four nylon screws and nuts, trimmed to remove extra material. Similarly, attach the IMU to the top of the case using 4 nylon screws and nuts. The battery may be affixed to the top of the case with double-sided tape to avoid motion inside the case during data acquisition.

4. **Attaching to the Belt**

First cut strips of the grippy fabric to fit the width and length of a section of the belt between the clip and length adjuster. This fabric should be sewn to the interior of the belt in this location, so that it meets the skin to prevent slipping during use. The fabric of the belt is very thick; sewing machines were not able to maintain a stitch in the fabric, so hand sewing is recommended. Next, cut two small strips of Velcro (one hook and one loop strip). These only need to be a few inches long. Attach one side to the exterior of the belt, where it would sit in the center of the participants back. This will be in the center of the section where the grippy fabric is sewn, but on the other face of the belt fabric. Attach the other Velcro to the center of the case bottom. The device can now be attached to the belt via these two Velcro strips and the belt can be attached to the subject.

VIII. HUZZAH CODE

All code can be found on the official project GitHub page at https://github.com/aw58/HMSU2.git. The code uploaded to the Huzzah is written as a .ino file for use with the Arduino Integrated Development Environment (IDE). Instructions for setting up the Arduino IDE can be found in the software setup section of this document.

The code uploaded to the Huzzah serves two purposes: gather the data and send it to an IP address via a serial connection. For a connection over Wi-Fi, please see "Huzzah_wifi_send_game_and_accel.ino"

The data being sent includes the Game Rotation Vector, in the format of a quaternion, as well as the x, y, and z values for linear acceleration. The total format of the sent data is:

"GR_w, GR_i, GR_j, GR_k, A_x, A_y, A_z\n"

where "\n" is a newline, printed as Serial.println by the Huzzah.

The Game Rotation Vector does not utilize any data from the magnetometer and is uncalibrated with respect to the Earth's magnetic field. This choice was made because the calibration causes discontinuities in the data, leading to more accuracy, but less smoothness for the researchers and the subject viewing the cursor. To avoid such anomalies, the Game Rotation Vector was chosen.

Additionally, Linear Acceleration data in all three directions is used for an approximation of positioning based on integration. Linear Acceleration from the IMU, as opposed to raw acceleration values, has gravity already subtracted; this format reduces the mathematics on the side of the host computer to calculate positioning.

For additional data which can be sent from the BNO085 IMU, please see the BNO085 data sheet as well as its library for accessible methods and data names here.

i. Wi-Fi connection and control

The primary purpose of connecting the HUZZAH and Unity to Wi-Fi was for an overall faster transmission of data between the devices. Early within the design process we discovered our initial idea of using Bluetooth Low Energy (BLE) for data transmission proved to have a significantly lower sampling speed than expected and required for the project. Data transfer over BLE reached a maximum at around 60Hz per sample of data that was being sent, but this does not give us enough samples per frame within Unity to run a VR environment efficiently or work at the standard that we were given for the project. Therefore, Wi-Fi was deemed a more viable option.

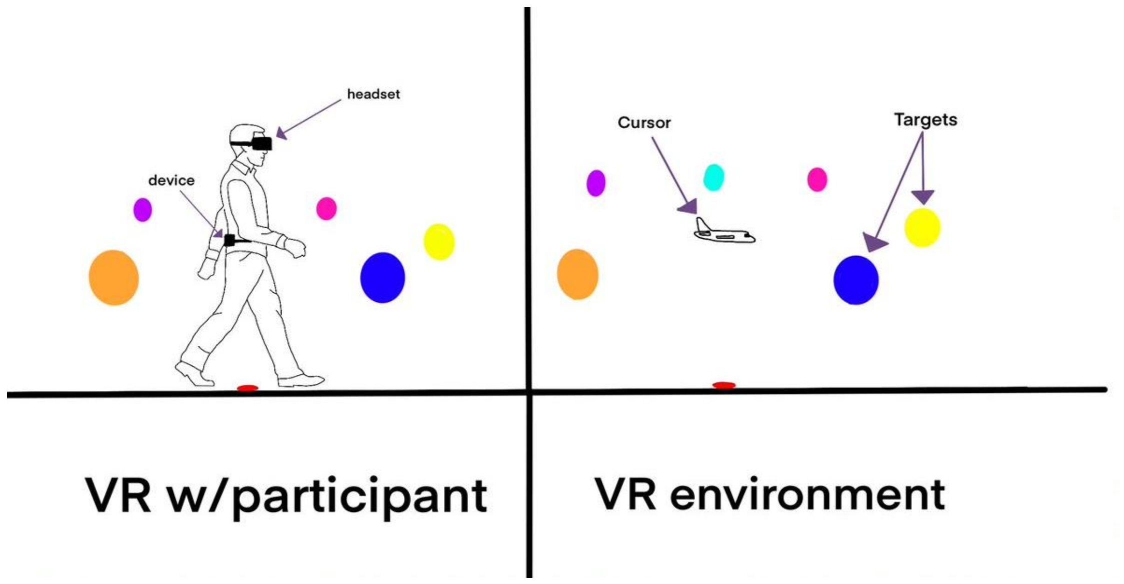
The basic control of the Wi-Fi was set up in the HUZZAH and in Unity3D. To set up a wireless data transmission network, we looked to connect the HUZZAH to the Wi-Fi of the available network. The device had to connect to the network's gateway (router) and establish an IP for the HUZZAH to be the server. This was accomplished by code in the Arduino IDE. We also initialized the IP address of the HUZZAH and created a port number so that it can be connected to the network. The data output from the Huzzah can be sent to the server client (Unity) in the same format as it would utilizing the serial port, but with more initialization and code on the client (Unity) side.

For Unity, we had to establish a client connection with the server on the HUZZAH so that the data could be successfully transferred. This was done using two C# scripts: one establishing the connection and reading the data, and another that was attached to an empty game object so that we can control the incoming data. The script establishing the connection was completed through the TCP client library in C# and utilized multi-threaded programming to read data and check to see if the client is connected at the same time. The other C# script just initialized a client object from the class of the other script.

The results of Wi-Fi data transfer are promising, but it did not get finished for the deadline of the project. We were able to successfully send simple integer and string values over across the network to Unity at extremely high transfer rates, but once we began to try and send IMU data, it did not send properly. Our initial discovery shows that it is most likely an issue with the Huzzah code, in which writing to the client is somehow different than printing to the serial port, and that this causes data to not be collected. We were not able to confirm this suspicion.

IX. UNITY

The VR game for use with the Huzzah was developed in Unity, a game development engine that has multi-platform support and is an extremely powerful tool for furthering the development of the project. This section describes the game's structure and source code, as well as how to use the game application.



The game is made up of a VR camera rig which allows a player to view and move about in the space, target spheres to demonstrate functionality for testing purposes, and a cursor object which turns and moves according to the device. The spheres, when hit by the cursor, change color and make a sound to indicate they have been reached. Additionally, the cursor and targets are both in front of the player so they can move according to the testing environment they see. The game does not yet have the ability to reset the environment while playing.

i. Creating a new game

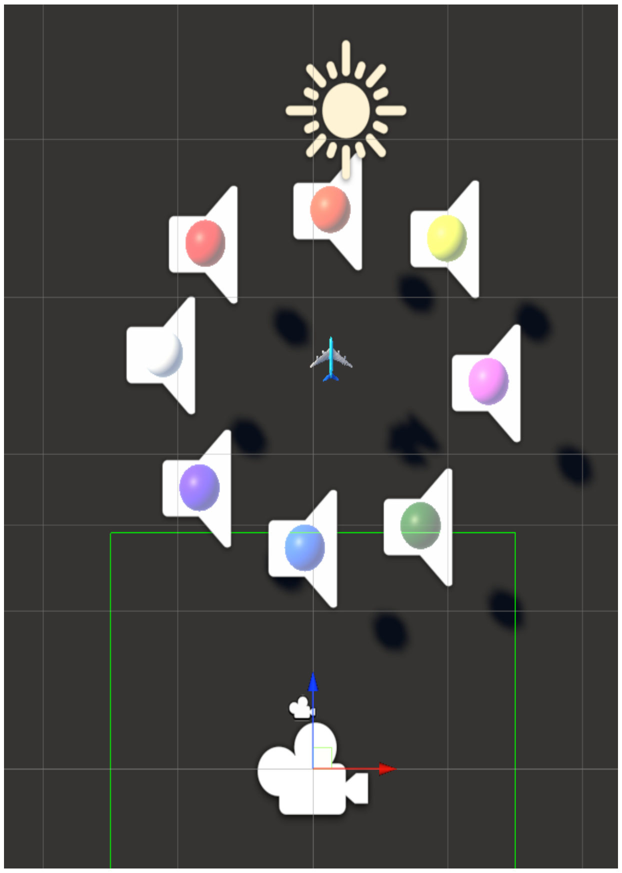
To create a new game, read the following notes, then watch this basic tutorial here for guidance. Notes on new game creation: the HMSU2 project was created in Unity 2020.3.3f1, but to create a new game, use whichever version of Unity is suggested for use at the time. When creating a project with the Universal Render Pipeline, the project comes with a sample scene; this can be deleted, and a new scene should be created. During the tutorial, you are tasked with creating your own XR rig from scratch; instead, you may do this by right clicking in the object manager, then right click >> XR >> Stationary XR Rig to create a default rig. You may deselect and hide the XR Controllers within the rig, as they are not necessary for the HMSU scene to operate. Additionally, the basic scene will only need the plane and XR rig from the tutorial, as all other objects are specific to the project.

Next, navigate to edit>project management>XR plugin management and select for which platforms you would like the game to be able to run on. If you would like the most versatility, choose at least the Oculus and Windows Mixed Reality packages

ii. Setting up the HMSU Trial Scene

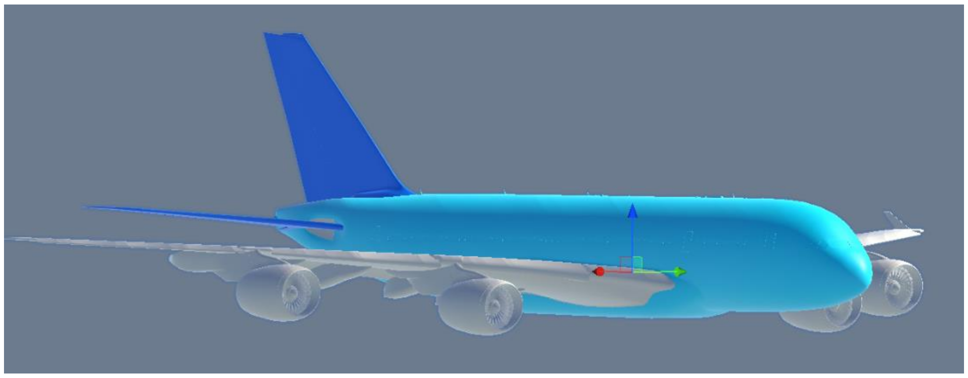
From the end of the tutorial in the section above, your scene should have a plane and an XR Rig. At any point during development of the scene, as long as a supported VR headset is connected to the computer and it is turned on, hitting play in the Unity environment will begin the game and the user can view the scene while wearing the headset.

First, make sure you have all the required C# scripts downloaded from the HMSU GitHub, and that you have moved these somewhere into your project for use. In the HMSU trial scene, these are placed in the "scripts" folder within the project's assets folder.



For creating the targets, create a new sphere (any shape will suffice) and place it in your game space. Then, add a "Rigid Body" component to the sphere; be sure to disable "gravity" for it so the sphere will float in place. Next, add a "Sphere collider" component. Again, any shape is appropriate, but matching the collider's shape to that of the visible object is best for reducing confusion for the patients. Finally, add the "Color Changer" script, attainable from the GitHub to the sphere. You may upload any audio clip into the "Collision Sound" slot, but a sample one may be found on the GitHub as well. Make sure the audio clip is somewhere within the project's assets folder, then you may click and drag it into the slot. The Audio Source is the location from which the sound will play, and this is most likely to be the object to which you have attached the Change Color. You may create new materials to color the sphere by right-clicking in the object manager, then selecting new>material and choosing the color you want. Finally, you can make copies of the sphere to fill the scene as you'd like.

For the cursor, an airplane is used to clearly show roll, pitch, and yaw to the participants and researchers. To begin, create an empty game object and label it Cursor Parent. The Cursor Parent should have the "Interface" script as a component. Then, make sure the Airplane object is in your Assets folder. Click and drag it into the item manager so it is a child of Cursor Parent.



The plane will be its original size and orientation; change the position to 0,0,0, the orientation to –90,0,90, and the scale to 0.0001 for each direction. This will place the plane in a manageable location for further manipulation. You may add materials to color different sections of the cursor to provide additional clarity for viewing its orientation in the game space.

Lastly, you may create a sphere which moves along with the plane and hides the plane's orientation to the subject. To create this, make a new sphere and place it as a child of the airplane object group (not the Cursor Parent) and add a rigid body (without gravity). The position of the Cursor Parent will automatically apply to the sphere. To hide or show the sphere, toggle the view in the column to the immediate left of the sphere's name.

iii. Unity Script Control

All Unity code is written in C#, Unity's native language, and the main script used in the trial scene is "Interface.cs." This script controls collecting and parsing data, writing to a file, any additional mathematics on the data, and finally, using that data to orient and move the cursor object in the scene. Another script called "Color Changer" handles changing the color of target objects and playing a sound to indicate success in reaching one. Color Changer is optional for full data functionality and is only used as a demonstration of capability for testing applications.

In addition, the code contains abundant comments on what each line does; please see the code itself for an explanation of what each line does. This document will cover the overarching use of that code.

In Interface.cs, a serial port object is initialized which then reads from a COM Port on the computer. Note that the number of this COM port must be updated with the number your Huzzah connects to, visible in either your computer's "Device Manager" or in the Arduino IDE's "Port" selection drop-down menu.

Next, after reading in data as strings from the COM Port, they are parsed into an array and filtered out (filtration was experimentally found and may be different per your system).

The data from the Game Rotation quaternion is used to create a new quaternion representing the orientation of the cursor. Before initialization, the cursor shows the orientation of the raw values with respect to the world-space axes. After the body-initialization period elapses, is relative to the original orientation of the Cursor Parent relative to the scene's world-space axes.

More importantly, the change in orientation of the cursor relative to the turning of the Huzzah is not guaranteed to be in sync. A pitch movement for the device may correlate to a yaw movement for the cursor. This is adjusted for by changing which components of the Game Rotation are assigned to which components of the transform.rotation update. The current orientation initialization design only compensates for rotating the orientation back to a home, but rotations after that are governed by the incoming data. Furthermore, if the plane seems to be rotating about a point which is not its center, that means the center of the airplane does not align with the center of the Cursor Parent; when initially placed on the same point in the game space, the cursor will rotate about a central point.

Positioning begins from the starting location of the cursor within the game space, as defined as where the Cursor Parent is originally placed in the game. All positions are then relative to that starting point. Position is calculated as a simple integration of acceleration and is meant to be a proof of concept with more mathematics and processing needed for accurate movement.

There are two waiting periods before which orientation and positioning data is initialized. The first of which, Table Initialization, is when the device is not moving. This allows the calculation of a corrective velocity bias. The second initialization period, Body Initialization, initializes the orientation of the device in the game space, as well as its starting position. The time when these occur is controlled by public variables which simply count the frames since starting the game before initialization.

X. CHANGING AND UPDATING THE EXISTING SYSTEM

i. Sending Alternate Data

To view the additional data that can be output by the Huzzah, please see the data sheet for the BNO085, as listed in the Appendix, as well as its supporting links for the methods required to access the data.

The Game Rotation Vector particularly does not use any data from the magnetometer. This is used for the purposes of smoothing out the data, which can jump around as corrections occur based on the magnetometer. For the most accurate orientation possible, use the “Rotation Vector” which also takes magnetometer into account. This may cause discontinuous cursor motion which may disorient the test subject relative to the targets and may be confusing when reviewing the data.

ii. Changing the Unity Environment

The sample Unity environment is set up with the basics of functionality and is intended to be the foundation of further development toward specific tests and applications. The scene itself should be altered for particular tests using the scripts developed by the team; however, the Interface and Change Color scripts provide basic functionality, and any changes, as opposed to additions, may cause issues.

Critical changes that are most likely to be needed upon starting to use the system include changing the COM port and adjusting the Huzzah to Cursor orientation in Interface. Exporting the game means these scripts will not be updateable; exporting should only be done for when the environment will be run with the same settings for long periods of time.

XI. ISSUES AND AREAS FOR IMPROVEMENT

Although many design goals of this project were met, there are still significant areas to be improved upon. One of the biggest areas for improvement would be finalization of data transfer over Wi-Fi. Using a wireless mode of data transmission would make this product more robust and better suited for its purpose of human movement and balance studies. The use of Wi-Fi could also improve data transfer rates. In addition, a better indicator of the position of the cursor/device in the VR game space would be beneficial to researchers, as this provides an opportunity for a larger range of applications. Although basic positioning has been established, there is still a significant amount of drift over time and mathematical error that accumulates quickly, which should be addressed before implementation.

XII. APPENDIX

GitHub Repository: https://github.com/aw58/HMSU2.git

Unity Project Zip file link: https://pitt-my.sharepoint.com/:u:/g/personal/aaw58_pitt_edu/EerNpHu7o6ZLvB9tCeGkgX0BTBuhmSAYk2-JKrGQt5WVmQ?e=sogeyq

Links to assist with future coding:

I. Adafruit BNO085

- Datasheet: https://www.ceva-dsp.com/wp-content/uploads/2019/10/BNO080_085-Datasheet.pdf
- Library User Guide: https://github.com/hcrest/bno080-driver/blob/master/UserGuide.pdf
- Wiring and Setup: https://learn.adafruit.com/adafruit-9-dof-orientation-imu-fusion-breakout-bno085/arduino

II. Adafruit Huzzah

- Datasheet: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-feather-huzzah-esp8266.pdf
- Wi-Fi Set Up: https://learn.adafruit.com/adafruit-feather-huzzah-esp8266/using-nodemcu-lua

III. Basic Positioning and Orientation via Unity

- https://docs.unity3d.com/ScriptReference/Transform.html
- https://docs.unity3d.com/ScriptReference/Transform.Rotate.html
- https://docs.unity3d.com/ScriptReference/Transform-position.html
- https://docs.unity3d.com/ScriptReference/Quaternion.html

IV. Orientation and Positioning Mathematics and Concepts

- https://ieeexplore.ieee.org/document/5371807
- http://www.chrobotics.com/library/accel-position-velocity

V. Coding Examples and Tools

- Unity setup: https://www.youtube.com/watch?v=gGYtahQjmWQ
- Writing to a text file: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/file-system/how-to-write-to-a-text-file
- Importing a Unity .zip file: https://gamedevbeginner.com/how-to-move-or-copy-a-unity-project-without-breaking-it/#:~:text=It's%20most%20relevant%20if%20you,)%20or%20Compress%20(Mac).