# Developing a virtual environment to study the effects of changing perceived sway on human balance


## Ludovico Attuoni - 1945949

Computer Science Project.
Supervisor: Dr. Sang-Hoon Yeo



**Word Count: 6860**

**Abstract**

This project serves as both an investigation of the relationship between the sway of visual stimuli and human balance, as well as the creation of an application that can be used to investigate this concept further, and explore other potential questions around the topic.

Using a virtual reality environment and a Nintendo Wii Fit Balance Board, 3 participants' postural sway and centre of pressure were captured while subject to augmented visual sway. The results suggest an exponential positive correlation between postural sway and augmented visual sway.

The product has been designed with lots of forward planning and easy customization in mind, so that non-computer science / VR developers can easily make large modifications to the trials. The code itself has also been designed to be easily alterable for future developers by using clear and straightforward code structures.

# Contents

# List of Figures

# List of Tables



# List of Algorithms

# Part I
# Introduction

## 1 Introduction to the problem

In the research area of postural sway, a key question to still be answered is to how exactly does the brain understands in real time the position and orientation (pose) of the body, and maintaining its balance. It is unclear how much weight the brain puts on various available senses to determine how it should adjust its pose to remain balanced. In this project I aim to explore the effect of visual cues of sway have on balance, by altering the visual reaction of moving head. When a person moves or sways, if their head moves, visual stimuli provide information that they are in motion. These stimuli move in an opposite direction to the movement. If a person turns their head right, visual stimuli move left across their field of visual. If a person moves their head forward, visual stimuli appear to move backwards, towards them. The size of this translation gives an observer information on how far their movement was. The perceived "motion" of visual stimuli is equal in size to the motion of the head. So how important is this relationship between action and visual stimuli?

The hypothesis is that, by changing the size of translation in visual stimuli which the user observers when moving, their balance and postural sway would be negatively affected. If a person moves their head forward while maintaining balance, but the objects in the vision travel faster and further towards the person, this might provoke compensation, as the person might believe they have swayed further forward than intended, and attempt to correct this by making another movement, despite already being balanced.

For a study to be effective, I would need to involve many participants. However, I have instead set the goal to make the product of this project a tool that can (and hopefully will!) be used in studies to come. To do this I will aim to create robust and clear documentation for using my software, as well as building it from source, and as well as documenting how the code is structured so that other future developers can augment it. I will also endeavour to make the project somewhat future-proof by avoiding brand-specific technologies and libraries. I will also try to make the project, without any more programming, able to test more configurations than just the ones tested within this report, in order to allow the product to be used to perform multiple different studies. This report and the small sample of data collected for it serves as a proof of concept for the product as well as an incentive to delve further into the research.

## 2 Related work

This project was inspired by and in some ways a continuation of a previous Computer Science research project ([1]). In that paper, the author discussed and demonstrated the efficacy of using a VR environment coupled with a force plate to measure the effects of visual stimuli on postural sway. The conclusion to this paper was that the VR offers a

lot of benefits and advantages over 2D representations.

There has been various attempts at measuring visual sway's affect on postural sway with simpler techniques. In another study ([2]), user's postural sway was measured by only using centre of pressure (force plate). The limitation here is that centre of pressure is not always a clear indication of postural sway, and more data tracking would be very beneficial.

# Part II
# Implementation

## 3 Justification of hardware choices

Firstly, to create an environment where I could manipulate users vision, a Virtual Reality (VR) environment felt like the most obvious choice over using a 2D display, due to the high fidelity in reactions to postural sway.

Secondly, to measure sway, I decided to use the position and orientation of the VR headset, as well as a force plate. The force plate would give a representation for the centre of pressure, which can be useful to determine a user's balance. I also used the 3D position and orientation of the headset.

### 3.1 Pimax 5k VR Headset

The 2 most important parameters for finding a suitable VR headset are: movement tracking capability, and a high Field of View.

Of the commercially available headsets, the majority of these fulfil the first parameter. I created a short list of VR products sorted by Field of View to compare potential options.
We ended up selecting the Pimax 5k, as it offered a good balance between high Field of View and price, after ensuring it's compatibility with Unity.

However for early implementations, I used my personal Oculus Quest 2. This enabled me to easily work from home, and also meant that progress could be made while waiting for the Pimax 5k to arrive.

### 3.2 Nintendo Wii Fit Board

Instead of using a conventional Force Plate I decided to use a Nintendo Wii Fit Balance Board (Wii Board). The reasoning for this was that a Wii Board is portable, allowing me to do more work from home. Wii Boards have already been used in other research papers and have had multiple papers validating the accuracy and sensitivity of a Wii board ([3], [4], [5], [6]). However, to be certain that the Wii Board I had access to is up to par, I

| Headset | FoV Diagonal, ° | Price | Notes |
|---|---|---|---|
| Star One | 210 | £3'000+ | Highest FOV available. Very expensive. Overpriced compared to overall specs. |
| Pimax 8kx/8k Plus | 200 | £995/£650 | There are some reports of distortion at the edges of the FOV, only noticeable if you keep your head straight and just turn your eyes. Reportedly just need to adjust the headset correctly to reduce/ remove distortion. This issue is said to be less noticeable than in the 5 series |
| Pimax 5k | 200 | £580 | Same FOV as 8k series and considerably more cost-efficient. There are some reports of distortion at the edges of the FOV, only noticeable if you keep your head straight and turn your eyes. Reportedly just need to adjust the headset correctly to reduce/ remove distortion |
| Index Valve | 130 | £459 | Considerably higher FoV than any other major VR headset not listed here. Very popular device - lots of support. |

Table 1: Shortlist of VR devices

conducted a few simple tests to ensure the sensors work as expected.

To do this I simply rested an object with a known weight (a 10lbs dumbbell) on each corner of the Wii Board, where the sensors are situated (Figure 1). I ensured to carry this test out on a hard floor so as not to skew the results due to the carpet dulling the reaction of the sensors. The results show that although each sensor correctly measures a change in weight when the dumbbell was placed upon them, it appears that they do not all rest at the same weight. Further investigation, by recording the sensor readings without any weight on, reveals a bias (Figure 2). However, the change in readings for each sensor from its value when no weight is applied to its reading when the weight is above the sensor is approximately equal to the 10lb weight. This implies that the sensors operate correctly but have a slight initial bias. As discussed in a paper measuring the effectiveness of Wii boards when compared to other devices ([7]), only relational values of the Wii Board readings should be used. (i.e. measuring change of sensor readings or comparing distance between sensor readings). It should also be noted that the bias is comparatively small, resulting in at most only 1kg of inaccuracy.

### 3.2.1 Related papers showing Wii Board validity

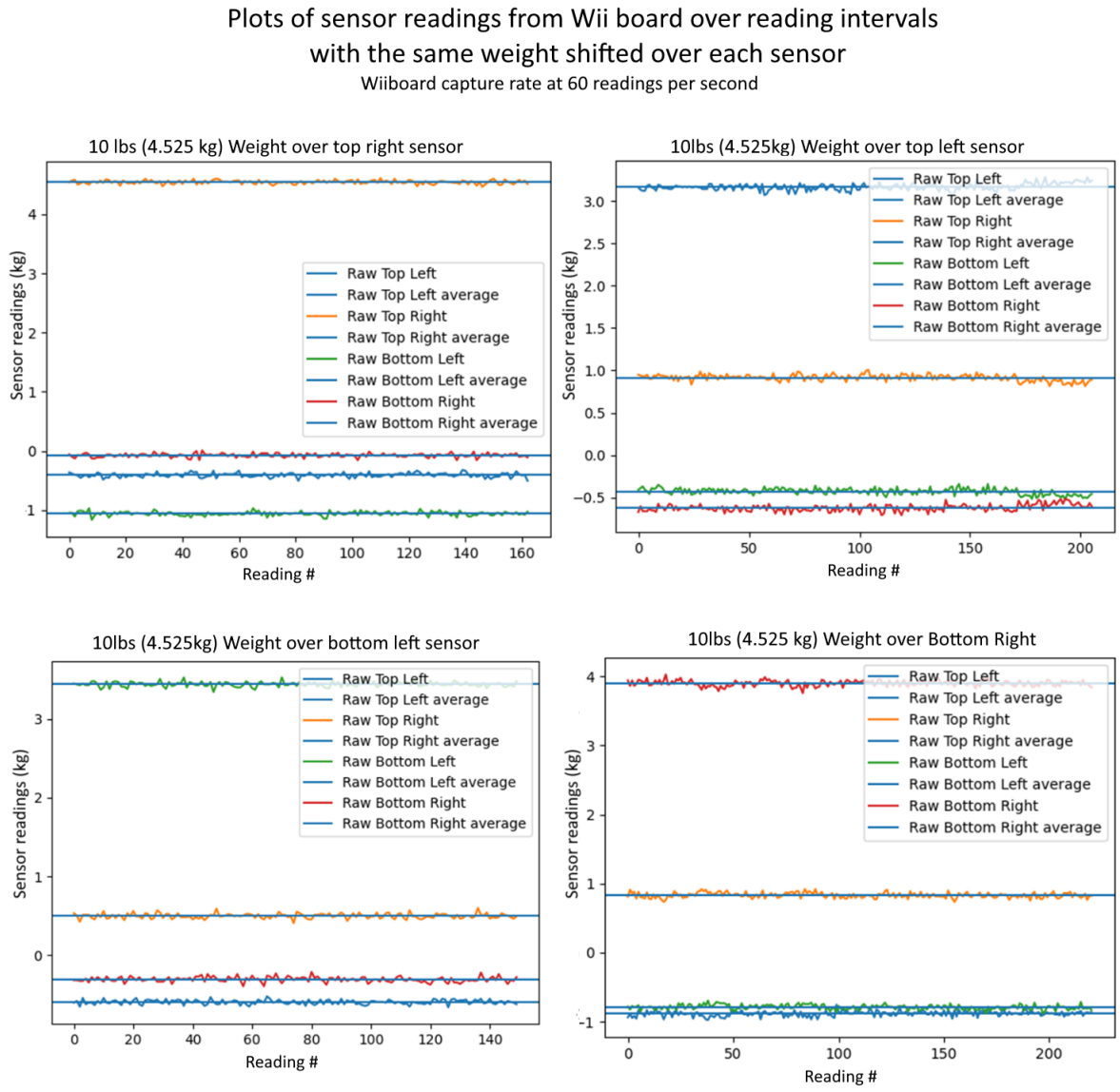



Figure 1: Plots to demonstrate sensors correctly identifying weight

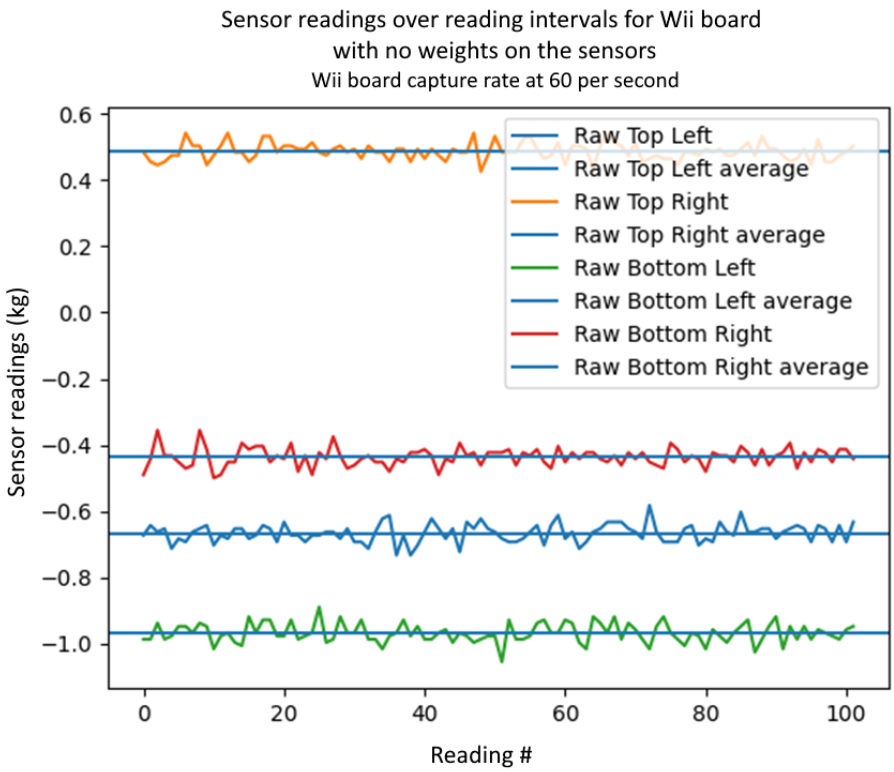



Figure 2: Control plot to demonstrate readings when no weight is applied

# 4 Justification of software choices

## 4.1 Unity v2020.3.1

In order to interact with a VR interface, as well as create a 3D environment, I decided to use the Unity game engine, using the latest stable version of Unity available. Unity has many advantages, as it offers a straightforward VR implementation. It has been used in various research projects ([1]), and as mentioned in one study ([8]) offers simple alterations to hyper-parameters to be done using the Unity GUI. Unity also helps handle many input actions and UI creation. This choice meant that much of the work in terms of rendering 3D graphics and interacting with VR headsets was readily can be managed with readily available packages. I decided to make use of the SteamVR[1] plugin for Unity [2], which not only makes use of OpenVR[3] (and therefore makes most PCVR devices compatible with my application). In addition SteamVR allows lots of control options that help users set up the Virtual Reality environment- in terms of setting up Base Stations and calibrating the headset.

[1] https://store.steampowered.com/app/250820/SteamVR/
[2] https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647
[3] https://github.com/ValveSoftware/openvr

## 4.2 Nintendo Wii Fit Balance Board - C-Sharp Socket Script

In order to connect the Wii Fit Board to my Unity project I decided to use the Wiimote C# library[4]. However due to a conflict in dependencies in the Wiimote Library and Unity, the engine I have opted to use, I decided to create make an external application with the Wiimote Library, which connects to the Wii Board and sends the data to the Unity project over a C# User Datagram Protocol (UDP) socket. I opted for UDP over Transmission Control Protocol (TCP) as arrival of every packet over this connection is less important than a reduced latency of data transmission. This implementation is common in server-client fast-paced games for the same reason. ([9]).

Once I had implemented this socket, I then needed to test whether the latency of data over the connection was acceptably low. To do this I created a "latency mode" on this external script which sends timestamps over the connection. The receiver in Unity then compares the timestamp received with the current time, and records the result (Figure 3). As shown by the graph, the average latency is at 0.2ms and peaks at 4ms.

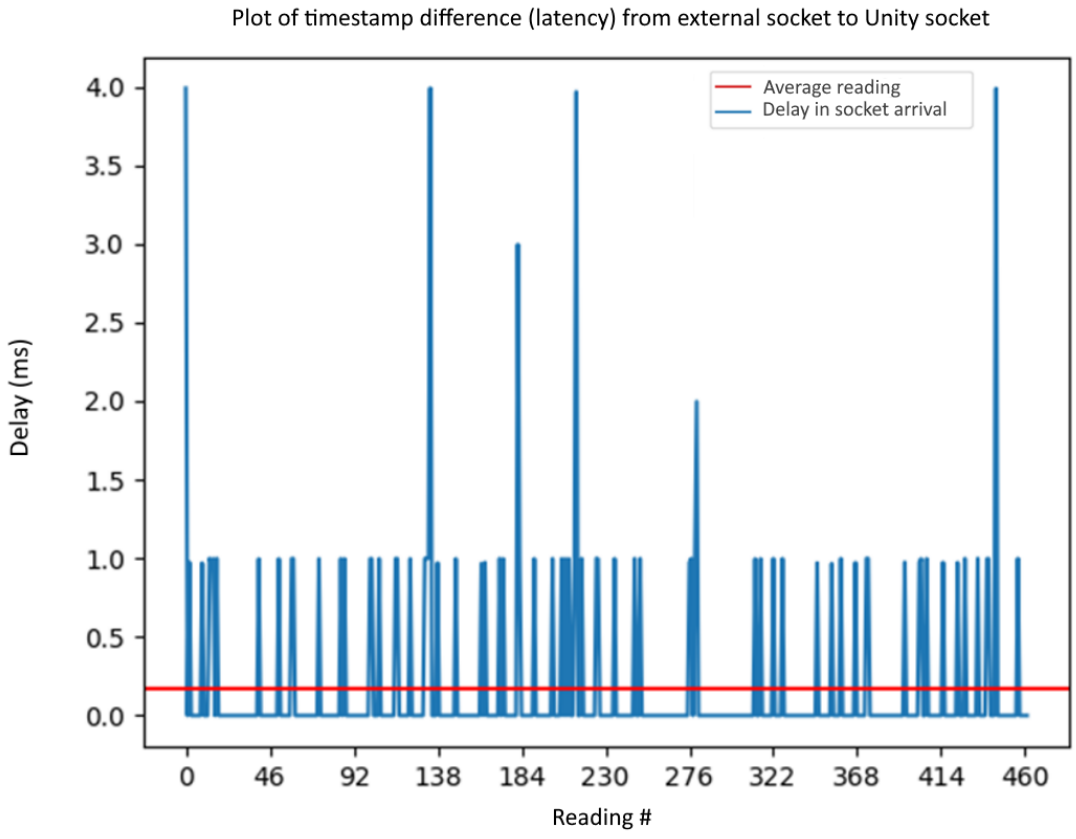



Figure 3: Plot of sockets latency

[4] https://github.com/BrianPeek/WiimoteLib

The other test I needed to carry out is the capture rate of the script, and the delay between an event and the corresponding reading. To do this I placed the Wii Board on a hard surface and struck a keyboard onto the bottom left corner of the board, so that the space bar was pressed. I then recorded the time of each space-bar press and compared to the record of the Wii Board readings (Figure 4). In the graph, any sensor reading peak that exceeds the threshold is considered a response to the space-bar. This final version of the script has a capture rate of 98% (when tested over many of these tests), and has a total delay 12ms with standard deviation of 5ms.

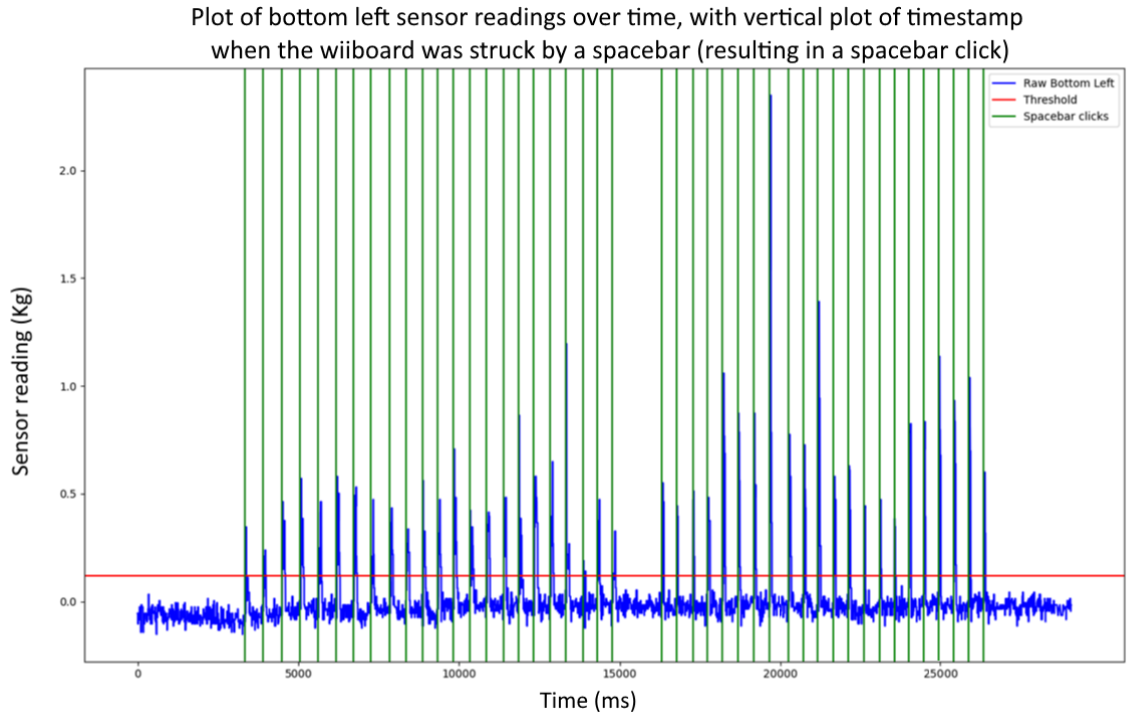



Figure 4: Plot of sensor readings latency

### 4.3 Python Matplotlib

I used Python with Numpy (NP) and Matplotlib (PLT) to create the graphs and figures. This is both because I am familiar with Python and PLT/NP, and also because these libraries allow for lots of customisation and flexibility on how to organise and display my data.

## 5 Software and Code Design

With the tools mentioned above, I created the system shown in Figure 5. This system overall works efficiently and is reasonably quick and easy to set up from scratch using my repositories. This system also allows for some flexibility in operating systems and set ups- one only

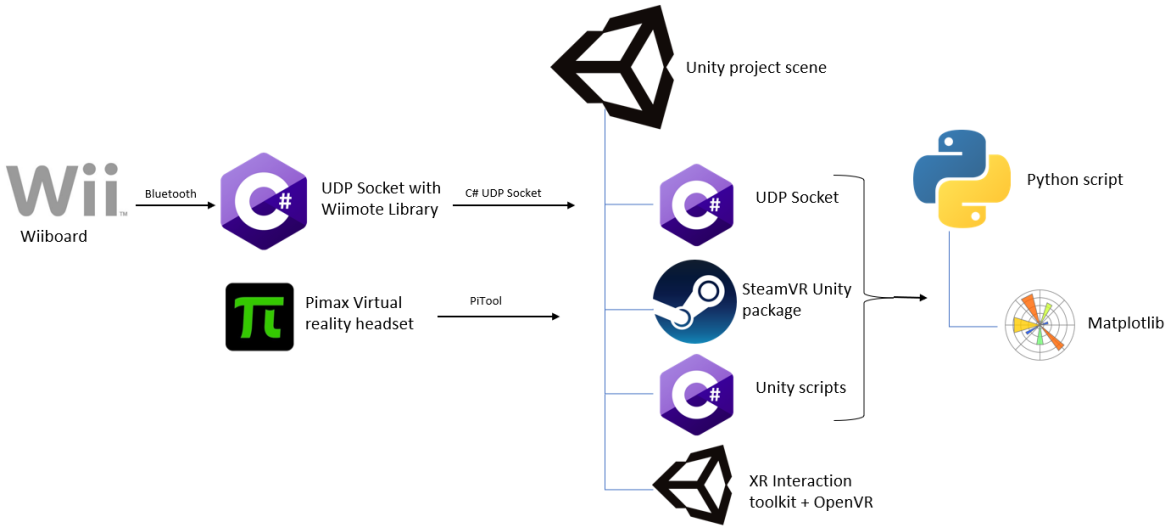



Figure 5: Hardware and software map

## 5.1 Nintendo Wii Fit Balance Board - C-Sharp Socket Script

### 5.1.1 Approach

The goal for this application is to enable communication between the Wii Board and the Unity program, with a secondary goal to provide some analytics and tests for the Wii Board performance. These goals, at a basic level, were achieved reasonably quickly and easily. I had already decided to use Wiimote library which made all communication and connection to the Wii Board reasonably straight forward, and all that was left was setting up a Socket that posts the data. The biggest challenge with this program however was finding a way to capture the data readings from the Wii Board as quickly as possible. My first implementation of this software used a NET System Timer object set to the lowest interval (time between scans) possible. The subsequent data about the Wii Board, when conducting the "Space bar test" (see justification for Wii Board) is as seen in Figure 6. Green bars indicate space bar clicks that correspond to peaks in sensor data, and red bars correspond to space bar clicks that have do not seem to correspond to any peaks. A peak is counted as corresponding if it is higher than the threshold (to cancel out any noise), within half a second of the space bar click, has not been associated another more likely space bar click. It appears that there are a substantial number of missing peaks in sensor data, (over several of these tests 60% of the space bar collisions were not captured), and there is a significant delay between the event and the reading ( 22ms delay with 23ms standard deviation). At first there were concerns that the Wii Board itself or the scan rate of the bluetooth connection was the reason for this delay and missing data. However, after some research on how the System Timer object operates, I realised it may have been the culprit. (include reason). Therefore I decided to switch from this to a while-loop with 0ms of sleep, running in a thread. This greatly improved the efficacy of the program and significantly reduced the delay and missing capture packets, as demonstrated in the justification for Wii Board in the report. Note also that the plot of the sensor readings

in Figure 4 has much more noise, signifying a much more accurate and higher-frequency reading of the Wii Board sensors, and highlights that the limitation was not the sensors themselves.

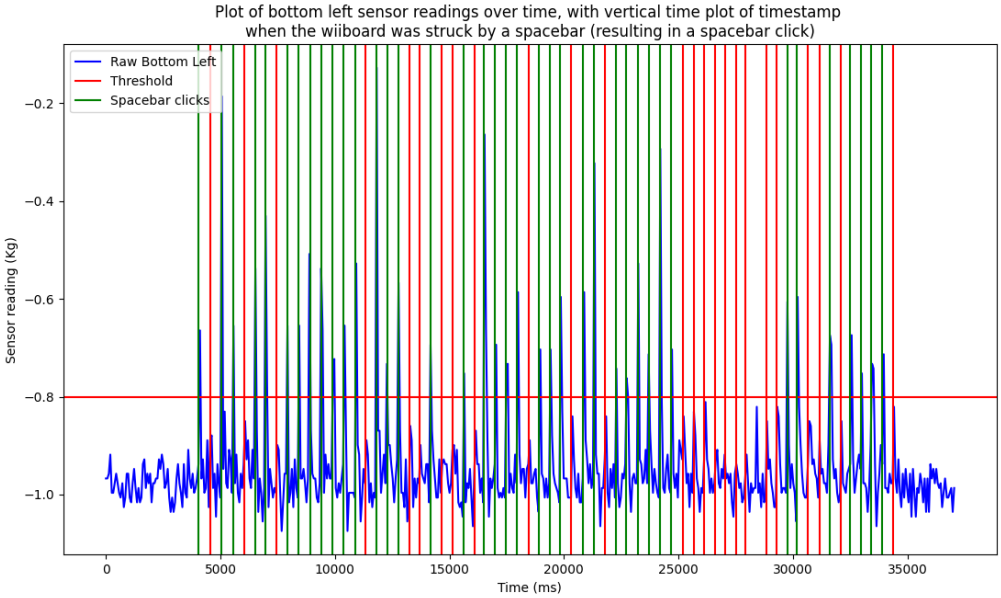



Figure 6: Socket script flowchart

To gain a full representation of how these scripts work, the following chart shows the flow of the program in reasonable detail. (Figure 16).

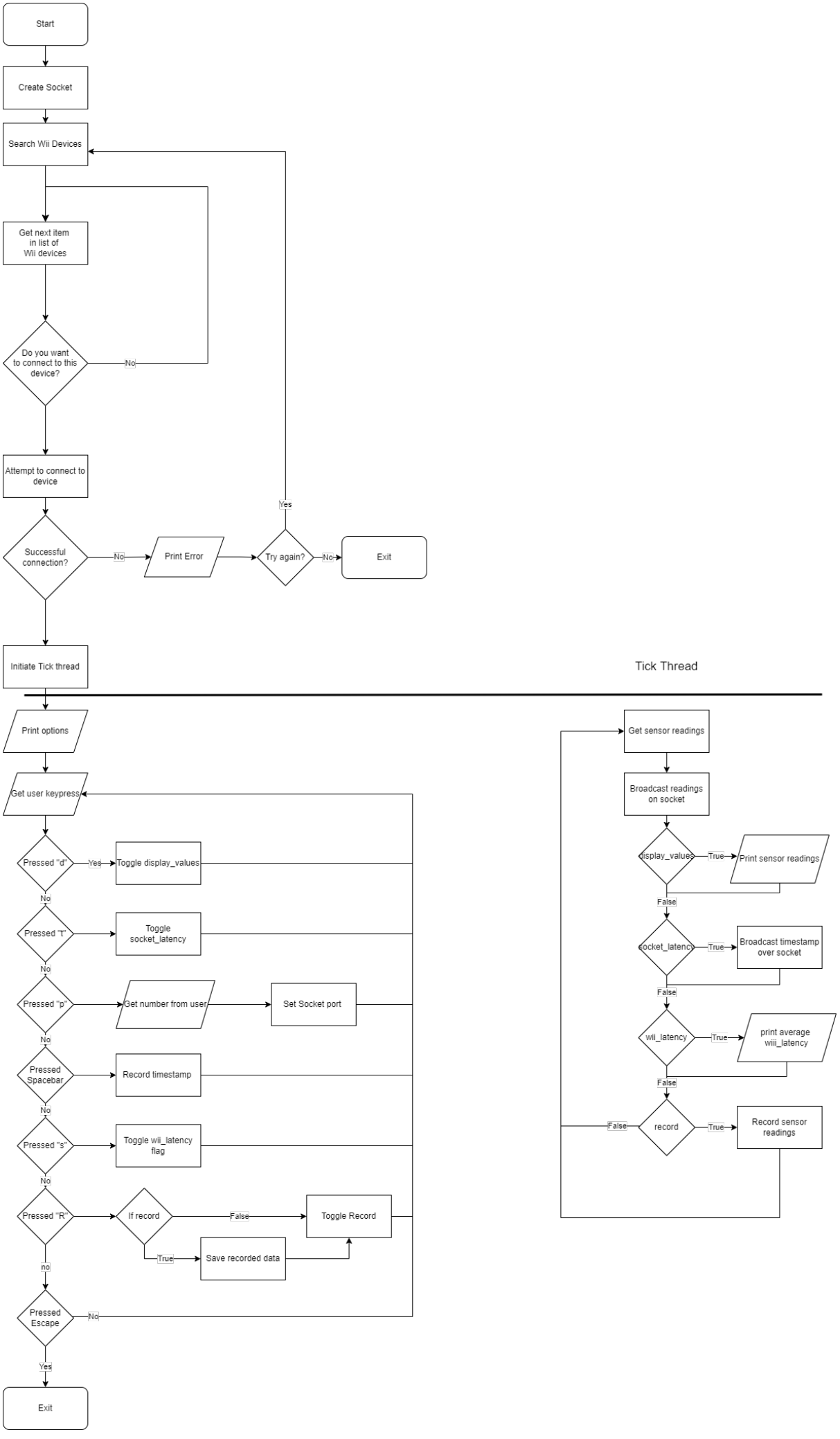



Figure 7: Flowchart of Socket Script control flow

### 5.1.2 Class Diagrams

Here are the class diagrams for the final implementation of this part of my code. It includes the Unity Socket which is only found in the Unity scene, not in this program, but has been added in the diagram for reference and clarity on the overall structure. The Unity UDPSocket inherits from the UDPSocket class, in order to ensure the code would all work predictably.

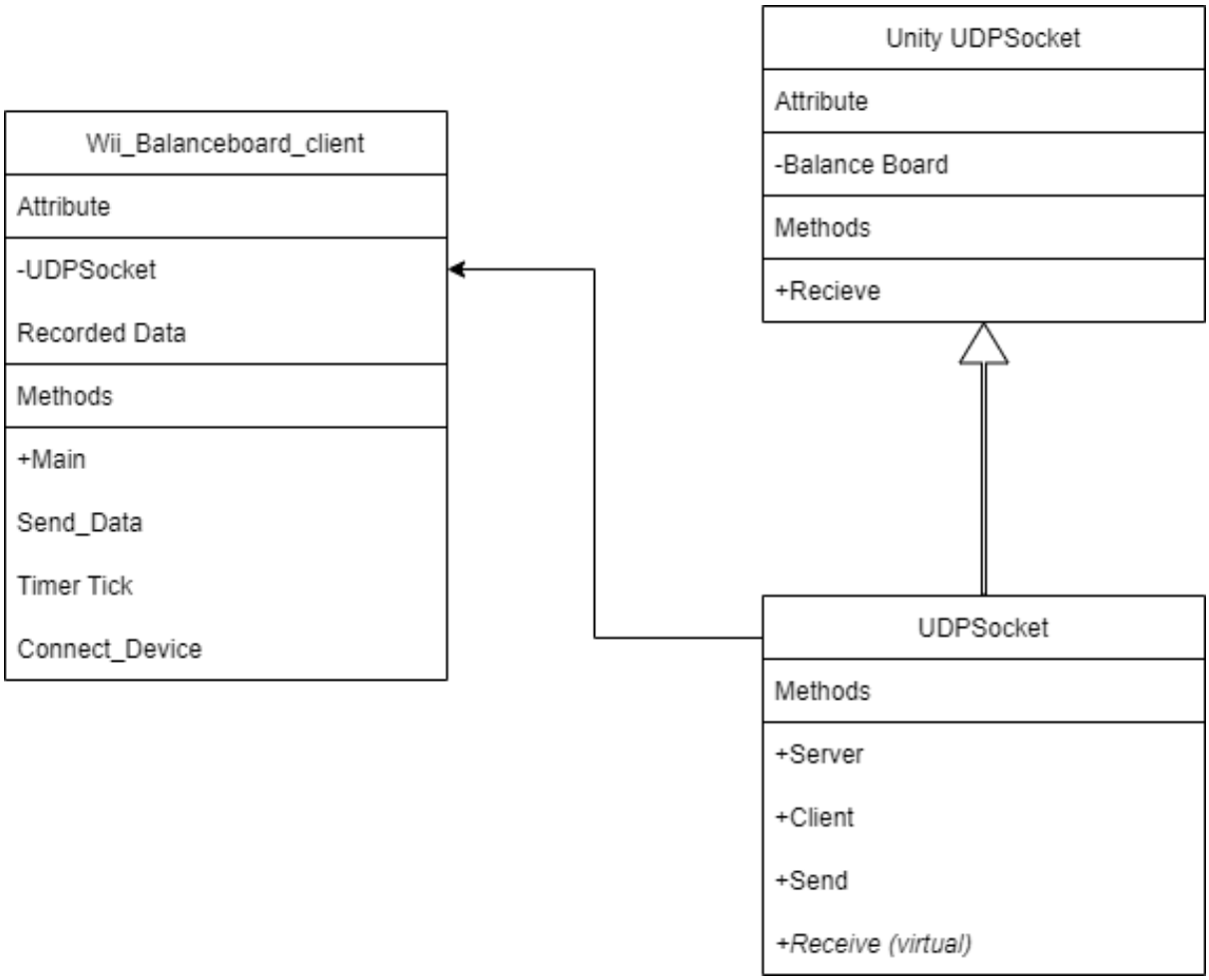



Figure 8: Class Diagram for C Sharp utility tool and UDP Socket

## 5.2 Unity Project

The overall aim for the Unity project is to create a virtual environment where, taking in the user's sway, the scene can simulate visual stimuli to move in a way that is proportional to the user's sway, but multiplied by some constant factor (Figure 9).

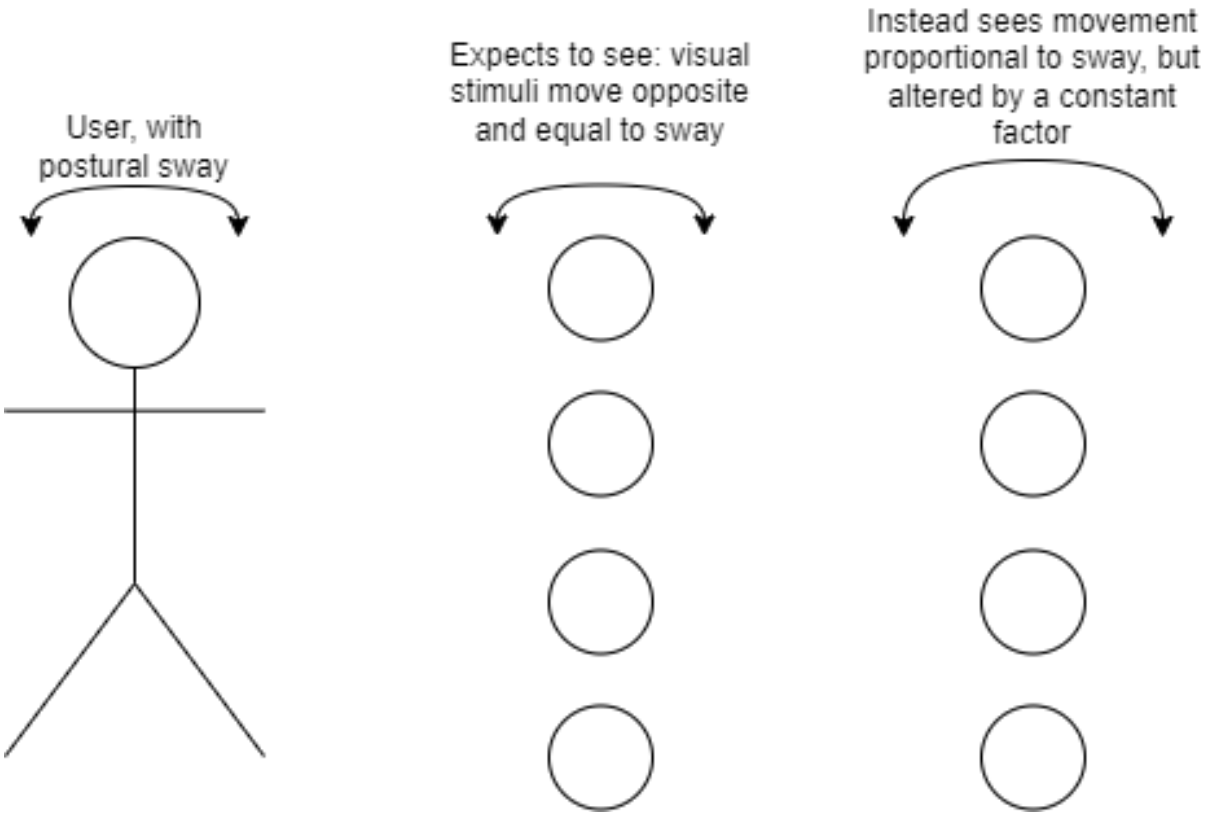



Figure 9: The concept of the Unity project

This portion of my project saw 3 distinct versions, each addressing the problem of the last. I have therefore separated my approach into 3 sections to explain the process of how I arrived to the final implementation. At every point of the journey, this program can be separated into 3 distinct sections, which I will address the progress of in each version.

1. **Creation of Visual Stimuli**
2. **Capturing Postural Sway**
3. **Augmenting Visual Sway**

### 5.2.1 Version 1

**Creation of Visual Stimuli**

For my first implementation of visual stimuli, I decided to simply create a tunnel of random small particles, with the intention of letting them scroll around the user. The code for this was reasonably simple, the following pseudeocode illustrates my method:

**Algorithm 1** Setting each particles position in a tunnel

**Require:** particle, minimum tunnel length, maximum tunnel length, minimum radius, maximum radius

$z \leftarrow$Random(minimum tunnel length, maximum tunnel length)
angle $\leftarrow$Random(0, $2\pi$)
radius $\leftarrow$Random(minimum radius, maximum radius)
$x \leftarrow$ radius $\times$ sin(angle)
$y \leftarrow$ radius $\times$ cos(angle)
particle.position $\leftarrow$ ( x , y , z )

By stripping the particles of colliders (collision detecting objects in Unity) and a few other non-essential components, I was easily able to render 10s of thousands without experiencing lag in VR. Another key optimisation I implemented was to create each particle GameObject as a child of a parent "Tunnel" GameObject. What this means is that, instead of having to move each particle individually based on user input, I can simply update the tunnel position, and Unity will instantly move all the particles in to it. This improves performance massively as well as making much cleaner and simpler code.

The issue with this is that the majority of the motion happens in the user's peripherals or outside of their vision altogether. The particles far in front of them in the tunnel hardly appear to move, as they are so far away.

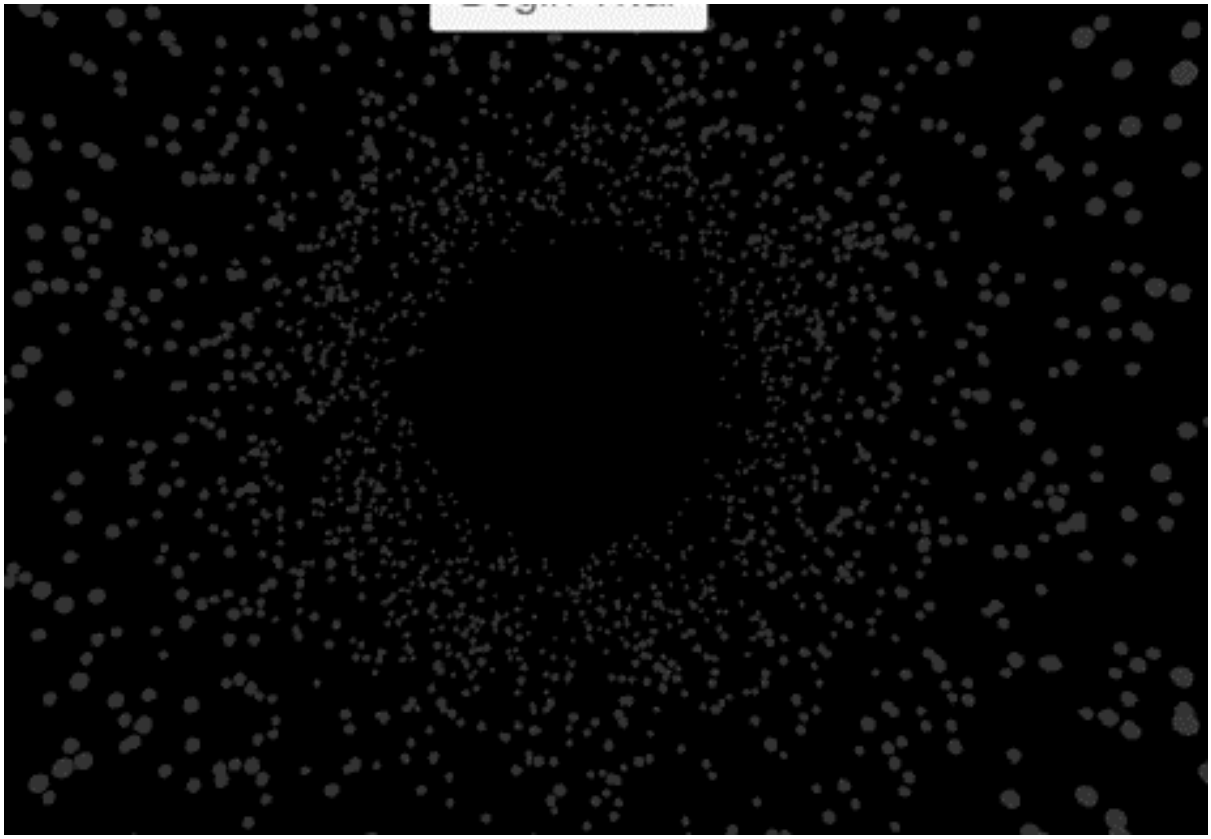


Figure 10: View of tunnel stimuli

**Capturing Postural Sway**

To capture sway, to begin with, I decided to use both the Wii Board and the Headset position - specifically in the z axis - to then move the stimuli accordingly. More specifically, I would take the frame-by-frame displacement of the headset in the forward direction, along with the change in difference between front and back readings of the Wii Board. The Wii Board data was also normalised based on the user's weight, to ensure a lighter person would not get lower values for the same sway as a heavier person.

The resulting sway capture quickly saw multiple shortcomings. Firstly, by capturing only 1 dimensional movement in sway, the complexity of sway was reduced too much. Moving at a diagonal or orthogonal to the forward axis would result in very little capture in sway, which quickly shattered any illusion that the visual stimuli were not being moved artificially. Secondly, the lack of angular velocity in sway capture meant that the visual stimuli could only move without angular velocity too, again not remaining faithful to what sway actually does look like, and also had a distinct lack of parallax which is arguably a key advantage to using a VR system over a 2D system. Both of these attributes suit the nature of the visual stimuli, but do not render a convincing visual sway to the user. Finally, by using the Wii board to capture sway, the received value for sway could reach very large values without the user ever moving their head. This is because the Wii board measures centre of pressure, so by simply periodically reapplying force on the toes one could sway the visual stimuli without moving.

To measure user sway for data analysis however, I simply record all 4 sensors of the Wii Board and X and Z Head positions, collected at 60Hz, together with corresponding time stamps to ensure accuracy. The resulting csv files are only 380KB in size, so there was no need to optimise or compress this data in any way.

**Augmenting Visual Sway**
As mentioned in **Creation of Visual Stimuli** and **Capturing Postural Sway**, this implementation was particularly 1 dimensional, meaning the augmentation was similarly very simple.

**Algorithm 2** Augmenting Visual Sway in 1 Dimension

**Require:** Wii Board sensors displacement, Head z position displacement

weighted centre of pressure ← Wii Board sensors displacement × Wii board modifier

weighted centre of mass ← Head z position displacement × head modifier

tunnel velocity ← weighted centre of pressure × weighted centre of mass
× sensitivity
× Time.deltaTime

The reasoning for having unique modifiers for both sway captures was to enable the two range of values to be lifted or lowered in importance, and to ensure that if one of the two has a much larger range of values, that I could reduce the impact by simply lowering it's modifier. The sensitivity parameter would then be the parameter I would change to test out different "amounts" of visual sway the user could experience, where 0 would be none, <0 would move the tunnel in an opposite direction to the expected visual sway, and a positive number would, move the tunnel in the expected direction. I also multiplied this value by Time.deltaTime, which is equal to the time between game updates. This is used to ensure consistency in movement, and to convert the units from "displacement per frame" to "displacement per second" which results in slightly larger and more manageable numbers.

### 5.2.2 Version 2

**Creation of Visual Stimuli**
Using the information learned from the previous version, this version featured a "cloud" of particles. In the spirit of keeping this portion of the code as simple as possible, so that I can easily process thousands of particles at a time, the algorithm is quite intuitive.

**Algorithm 3** Setting each particles position in a random cloud

**Require:** minimum radius, maximum radius

$z \leftarrow$ Random(minimum radius, maximum radius)
$x \leftarrow$ Random(minimum radius, maximum radius)
$y \leftarrow$ Random(minimum radius, maximum radius)
particle.position $\leftarrow$ ( x , y , z )

This was a considerable improvement from the last implementation. The result is a large hollow cube (hollowness defined by "minimum radius") consisting of random particles scattered randomly inside it. There were however a couple of issues with this implementation. Firstly, although the random coordinates are chosen from a uniform distribution, the resulting cloud of particles does not have a uniform number of particles in each quadrant. The result was that in some clouds, there might be lots more information (particles) in one corner vs an other. This inconsistency is not a major issue, but still something worth adjusting. Secondly, due to some efficiency measures made in the previous implementation, the particles did not cast or receive shadows, and light levels are constant across the whole scene. The result is that the particles are bright white no matter how far they are from the user, so particles in the distance that are half covered by closer particles just look like a part of the closer particle's body. These issues are illustrated in Figure 11.

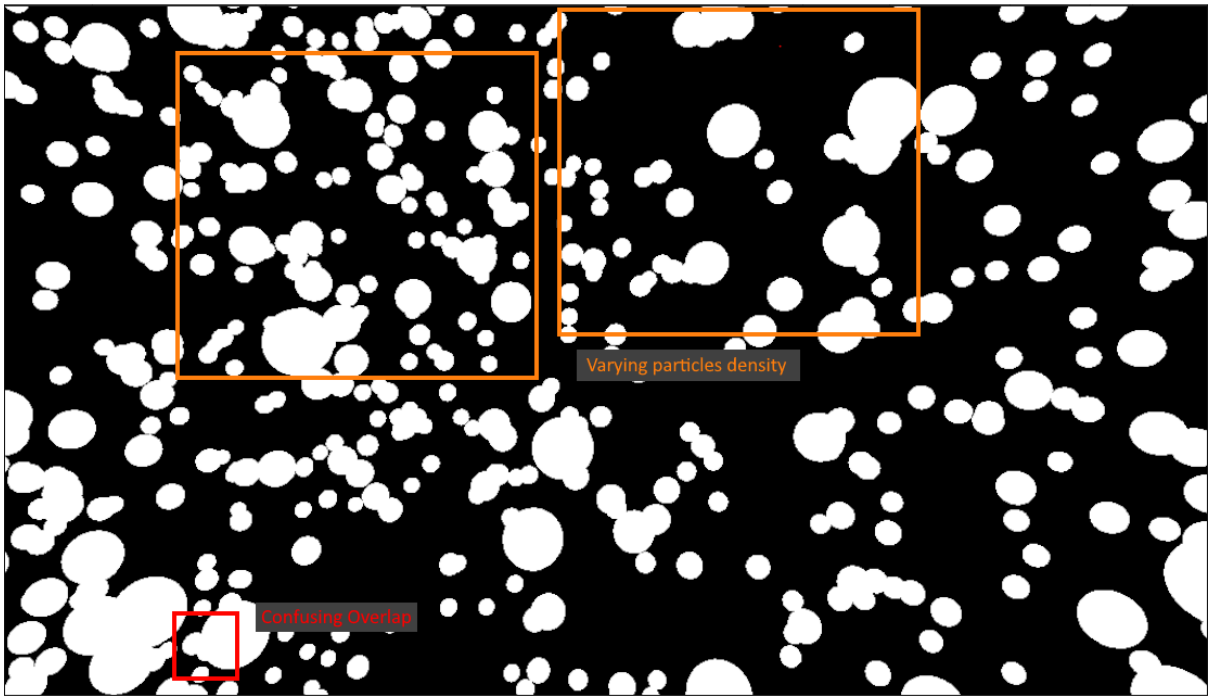



Figure 11: View of cloud particles

### Capturing Postural Sway

By leveraging the fact that the user would be always standing on the same spot (neces-

sitated by the Wii Board as a data source), we decided to model the user as an inverted pendulum. The camera GameObject is a child of a GameObject used to emulate this pendulum. By only capturing the displacement of head x and z position from every time stamp, I can then compute the angle at which the head position is at compared to a "base" point, at the bottom of the pendulum.

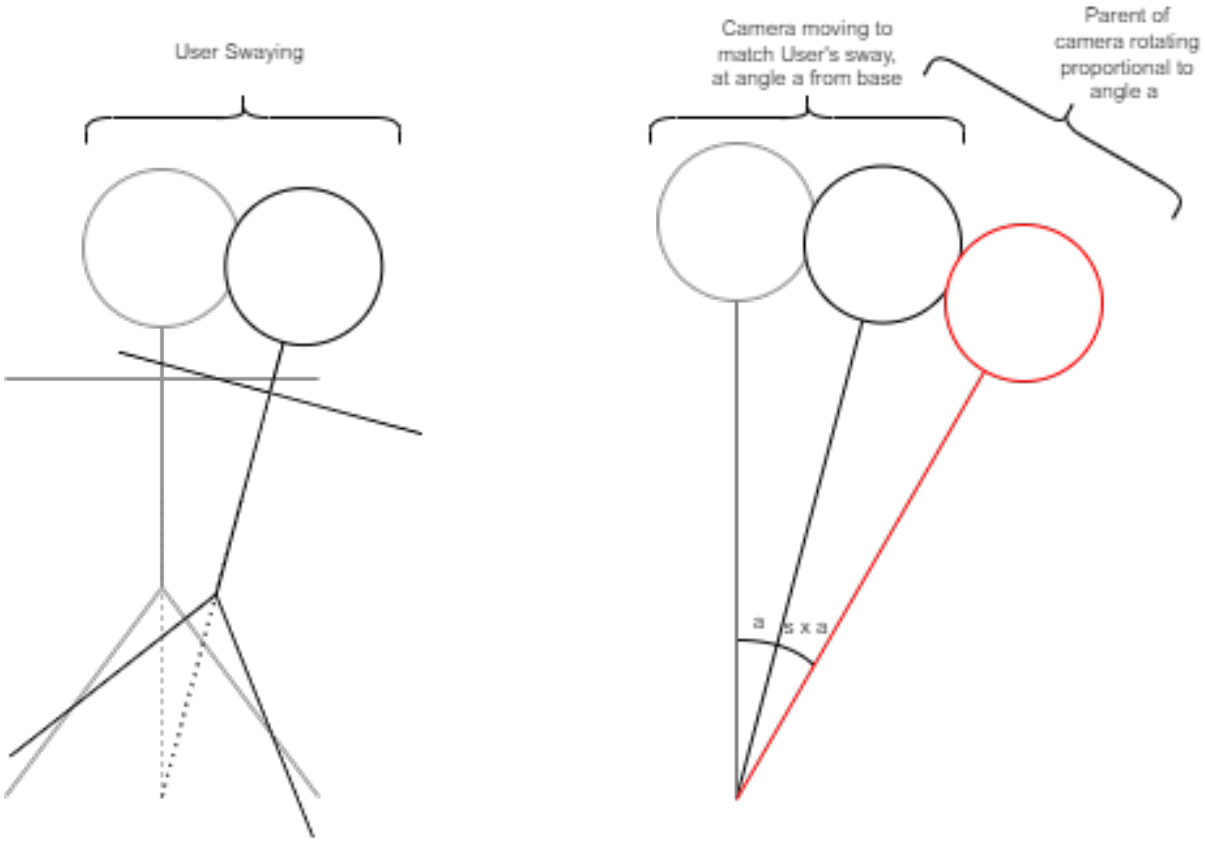



Figure 12: 2D Pendulum representation of user

This served as a surprisingly good estimation of user sway, however there are still some inaccuracies. Head turning for example was not captured, and furthermore many rotations and bends a person makes do no all originate from around their feet/ankles. The removal of the Wii Board for capturing postural sway to then augment visual stimuli however was a good improvement.

**Augmenting Visual Sway**

Using my new representation for the pendulum, augmenting sway was reasonably simple. Another advantage of using Unity was made apparent by the number of functions available for Vector-based mathematics. For example, SignedAngle returns a signed angle (-180, +180) between the two vectors, about a given axis.

**Algorithm 4** Augmenting Visual Sway in Pendulum model

**Require:** head position, previous head position, pivot position

pendulum vector ← pivot position - head position

previous pendulum vector ← pivot position - previous head position

axis ← pendulum vector × previous pendulum vector

angle ← SignedAngle(pendulum vector , previous pendulum vector , axis)
pendulum.RotateAround(pivot, (1 + angle) × sensitivity)

The resulting augmented sway was very off-balancing, and a great improvement on the last version. However, on high sensitivity values (2+) the fact that the user was simply rotating around a point and not augmenting their own sway was very visually obvious.

### 5.2.3 Version 3 (Final version)

**Creation of Visual Stimuli**

To overcome the issues of the previous version, I tweaked the cloud spawning code to ensure uniform distribution of particles. The idea is to make a cube of evenly spaced particles, then add some noise to the position of each particle.

Listing 1: Creation of Uniform random cloud of particles

```
1     async void makeUniformCloud(){
2         /*
3             creates a cube of particles, creates ''num_of_particles''
                  amount of particles, and uses this number
4             to determine how big the cloud will be
5
6             The algorithm creates a uniform cube of evenly spaced
                 particles in a cube formation,
7             applying random noise to each particle's position so that
                  they are off line
8         */
9
10          //number of particles per row
11          int p_row = (int)Mathf.Round((Mathf.Pow(num_of_particles, 1f
                / 3)));
12          // range which a particle can vary in x y z from its
                assigned position
13          float range = (max_radius)/p_row;
14          //number of particles on one face of the cube
15          int p_side = (int) (p_row * p_row);
16
17          for(int i = 0; i < num_of_particles; i++){
18
19              // each ''origin'' gets the coordinate for where the
                    particle is placed on the cube
20              int zorigin = (int) (i / p_side);
21              int zmod = (int) (i % p_side);
```

```
22              int yorigin = zmod == 0 ? 0 : (int) (zmod / p_row);
23              int xorigin = (int) (i % p_row);
24
25              // position (xorigin, yorigin, zorigin) would be the
                    particle's ''index'' position in the cube (think of it
                     like a 3d array),
26              //if the bottom left is (0,0,0), then (xorigin = 4,
                    yorigin = 2, zorigin = 2) would mean this particle
27              // would be the 4th from the left, second row from the
                    bottom and two ''faces'' deep into the cube
28
29              float x = 0, y = 0, z = 0;
30              // Reposition the (xorigin, yorigin, zorigin) to be
                    centred round the middle of the cube.
31              // (x = 0, y = 0, z = 0) would be the middle point in the
                     cube in all dimensions
32              x = (xorigin - p_row/2f)/ (p_row/2f);
33              y = (yorigin - p_row/2f)/ (p_row/2f);
34              z = (zorigin - p_row/2f)/ (p_row/2f);
35
36
37              // resize coordinates to desired cube size
38              x = (x * (max_radius));
39              y = (y * (max_radius));
40              z = (z * (max_radius));
41
42              // apply random noise
43              x = Random.Range(x , x + range);
44              y = Random.Range(y , y + range);
45              z = Random.Range(z , z + range);
46              //if this particle would be within the minimum radius,
                    skip creating this particle.
47              if(Mathf.Abs(x) < min_radius &&Mathf.Abs(y) < min_radius
                    &&Mathf.Abs(z) < min_radius)
48                  continue;
49              // Instantiate particle, set position and hideball
                    parameters
50              GameObject p = Instantiate(particle, Tunnel.transform);
51              Vector3 xyz = new Vector3(x , y ,z);
52
53              p.transform.localPosition = xyz;
54
55              p.GetComponent<hideball>().camera= camera;
56          }
57      }
```

To overcome the lack of shading, without a significant increase in rendering costs, I decided to implement a Unity feature called "fog", which adds shading based on the distance of the object from the camera. The farther the object is, the darker it appears. Figure 13

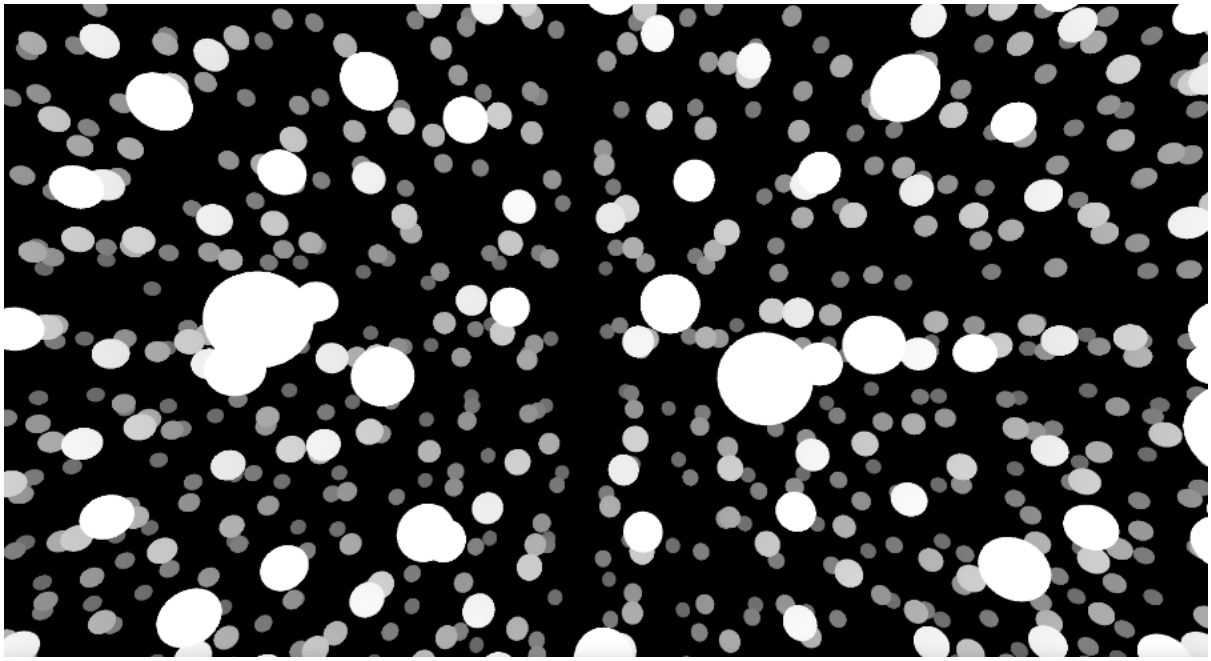


Figure 13: View of uniform cloud particles with fog

**Capturing Postural Sway**

As mentioned in the previous version, I needed to capture more accurately the orientation of the user. Instead of just capturing positions per frame, I now also capture the VR head set orientation, with a quaternion representation. The quaternion orientations stored by Unity describe the angle and axis that the object has been rotated by, relative to the starting orientation of the object. I capture the change in orientation using the following equation:

$$Q_{\text{Change}} = (Q_{\text{Previous}})^{-1} \times Q_{\text{Current}}$$

where $Q_{Previous}$ is the Quaternion representation of the previous frame's orientation, $Q_{\text{Current}}$ is the Quaternion representation of the current frame's orientation and $Q_{\text{Change}}$ is the Quaternion representation of the change between the orientations

**Augmenting Visual Sway**

To augment angular velocity in visual sway, in this version I decided to augment the amount which the change in orientation is effected each frame. By changing the orientation of the Unity camera by a factor of the change in VR headset orientation, I create a very convincing visual sway effect. The problem then was to reintroduce linear velocity. In order to apply the VR headset inputs to the camera it is not enough to simply capture the vector of movement of the VR headset and apply it to the camera. The reason for this is that the camera's reference frame is now different than that of the VR headset. For example, as illustrated in Figure 14, lets assume the sensitivity has been set to 1 (so every VR input is applied to the camera, then applied again, so doubled). Should the user their head turn 90 degrees then move their head forward 1 unit, the outcome of simply applying a doubled rotation and movement would result in the in-game camera rotating 180 degrees, but instead of moving "forward" it would move to it's left. The solution is to re-contextualise the displacement vector into the new orientation of the camera.

The final code for this section is as follows:

Listing 2: Augmenting Visual Sway with quaternion and vector dispacement

```
void Gain(){

        //rotation
        //get change in rotation
        var change_in_direction = Quaternion.Inverse(prev_rot) *
            real_world_input.localRotation;
        //get axis angle representation
        change_in_direction.ToAngleAxis(out var change_angle, out var
             change_axis);
        // apply gain and update rotation
        transform.rotation *= Quaternion.AngleAxis(change_angle * (1
            + sensitivity), change_axis);


        //get absolute change in position
        var displacement = real_world_input.position - prev_pos;
        // convert into vector relative to real world directions
        // ie if camera move in the direction it was facing, this
            would be (0,0,1)
        var unrotated_displacement = Quaternion.Inverse(
            real_world_input.localRotation) * displacement;
        //apply simulated camera rotation and gain
        transform.position += (transform.rotation *
            unrotated_displacement)*(1 + sensitivity);


    }
```

Note: the sensitivity parameter acts as an additive modifier to normal $1:1$ visual sway. The ratio for postural to visual sway is $1:(1+\text{sensitivity})$.
Two issues arose from this implementation however. One was user "drifting". This is when a user, when standing in the spot they start from, are no longer in the same spot in the VR environment. This can happen when, for example, a user leans forward during a period of augmented visual sway, then the period ends while the user is still leaning forward. If the user then leans back to their centred position, they will not travel as far back as they did forward. This might pose a problem as after multiple periods a user may end up at the edge of the particle cloud, therefore has a non-uniform distribution of visual stimuli. The solution was reasonably simple, a button was added to the UI than teleports the user back to the origin position. This can be used between trials. The second issue occurs when a user sways forward and breaches the minimum radius for the spawning particles, so the particles begin to obscure the user's view. The solution was a simple script on each particle which detects if it in the vicinity of the user, and hides if it is, and un-hides when it is far enough away.

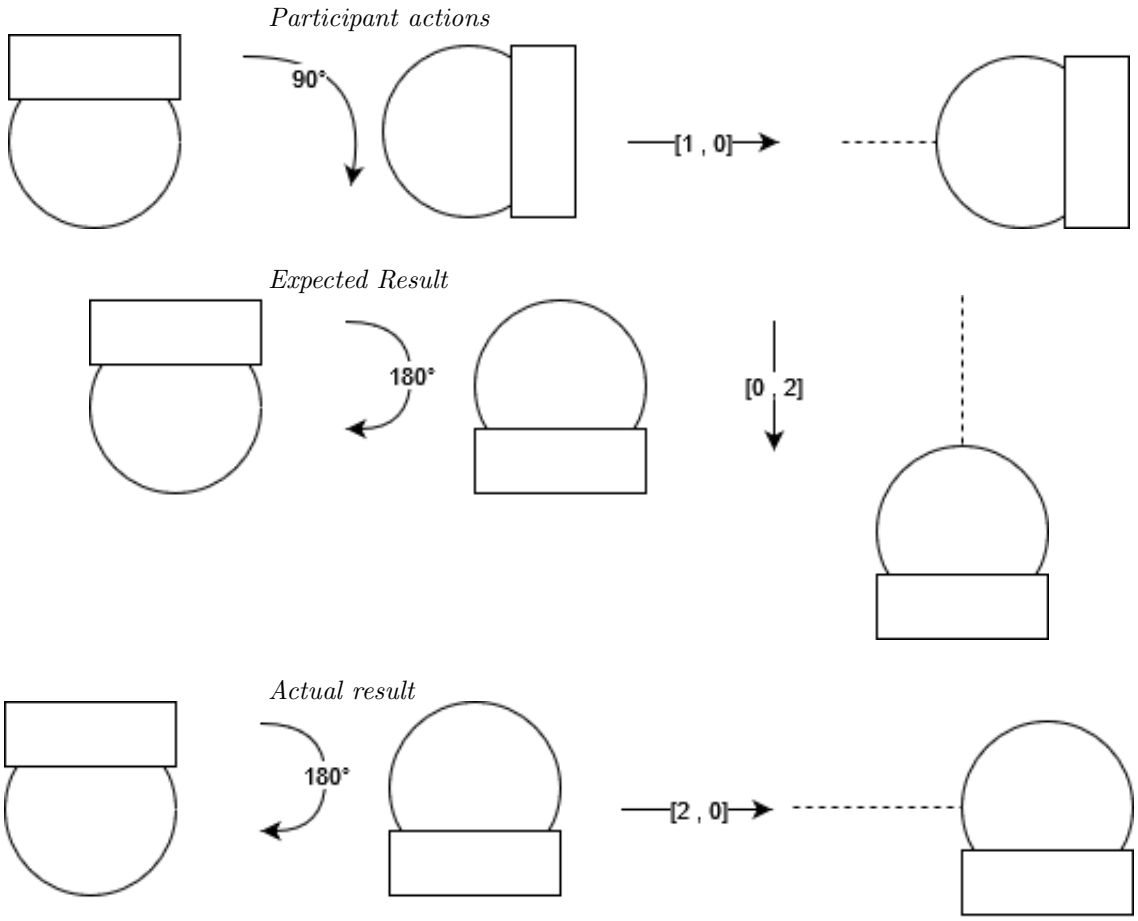



Figure 14: Problems occurring when applying position and orientation augmentation

### 5.2.4 UI

In order to allow users to easily change the sensitivity parameters, stage lengths and start and stop trials, I implemented a simple UI interface. Stages and trials are discussed further in the Method section. The UI allows future researchers to download the executable version of the project and run a variety of experiments, instead of building the project from source.

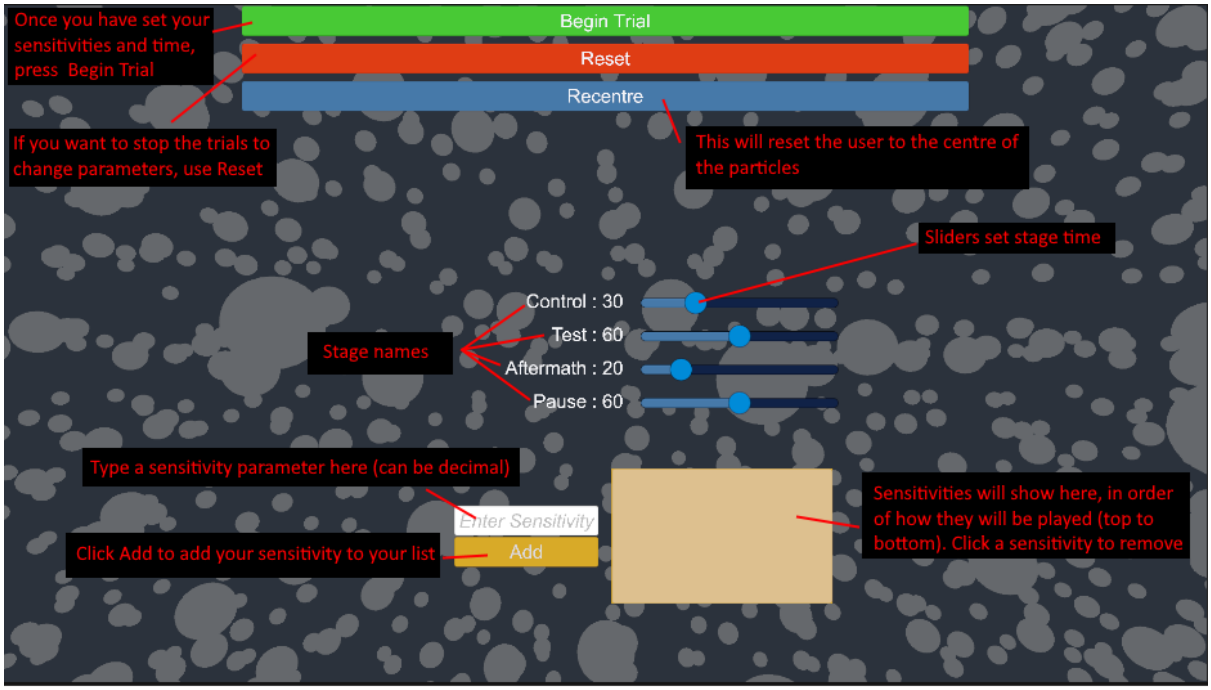



Figure 15: UI interface

To gain an overarching understanding of how the project operates, the following flowchart shows the program flow in reasonable detail, including the trials code.

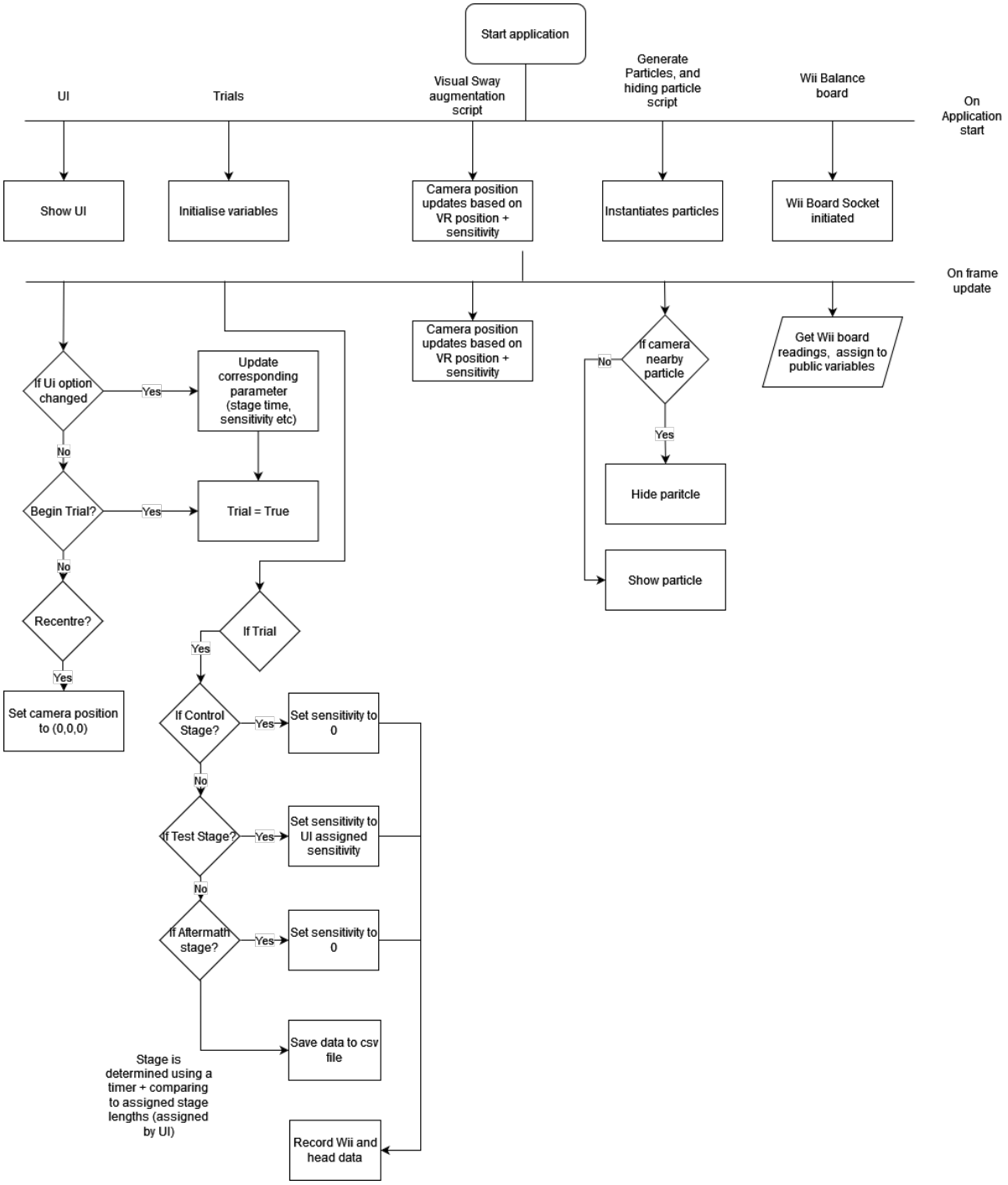



Figure 16: Flowchart of Unity project control flow

# Part III
# Experiments and Data Analysis

## 6 Aim

The aim of the experiments is to collect and analyse postural sway of people while visual sway is augmented. The experiments done for this report could only be done with 3 participants, so act only as an indication for what trends might be if conducted on a larger scale. The hope is that with what is learned from this set of experiments, an improved version can be implemented for a much larger sample. As discussed in the introduction, an overarching goal for this project has been to make a future-proof product that can be used and easily adapted to produce lots of valuable analysis on postural sway.

## 7 Method

The method for the experiments is to have a participant wearing the VR headset, and be subject to augmented visual sway. However, in order to instigate some initial postural sway, the participant stands on a curved foam pad. This offers slightly less stability than standing regularly, but considerably more stability than standing on one foot, which was found to be too unstable once subject to the augmented visual sway. In order to remove a bias from being unaccustomed to the VR environment, the participant is subject to an initial period (control stage) with unaltered visual sway. Following that is a longer period (test stage) of augmented sway and finally another short period (aftermath period) of unaugmented sway. The reason for this is to investigate the effect of changing visual sway back to an unaltered state, and seeing if this also has an effect on postural sway. Between testing each sensitivity, the participant also had a longer period (pause stage) without the headset on in order to reduce fatigue of being in the VR environment as well as try to mitigate bias for being too used to the VR environment before the next test (Table 3). I decided to test 6 degrees of augmented sway (Table 2), however I ensured the order in which the participants were tested was different and not in any specific order. This is to ensure no bias from expecting a certain amount of visual sway. The orders also different for each participant, in order to further remove this bias (Table 4). To ensure this lack of bias, the order in which each participant was tested on was entirely random.
The reason I selected these sensitivities is because I wanted a spread of sensitivities from only slightly noticeable (0.5) to undeniable (2). The reason I chose 0.5 intervals was to reduce the amount of trials, as each trial would take 2 minutes 50s. However I inserted 1.75 as while I was testing sensitivities, the jump from 1.5 to 2 felt much more dramatic than 1 to 1.5. I also included -0.5 to investigate the opposite effect in visual sway, where visual sway is less than real sway.
The participants were asked only to stand on the foam and not perform any actions. They were encouraged to look around the scene, however. This was to encourage some input to postural sway such that the visual sway can be augmented.

| | | | | | |
|---|---|---|---|---|---|
| -0.5 | 0.5 | 1 | 1.5 | 1.75 | 2 |

Table 2: Visual sway modifiers used in Experiments

*The values corresponds to a multiplier of the unaltered visual sway provoked by the postural sway*

| Participant | Trial 1 | Trial 2 | Trial 3 | Trial 4 | Trial 5 | Trial 6 |
|---|---|---|---|---|---|---|
| 1 | 0.5 | 2.0 | 1.0 | 1.75 | -0.5 | 1.5 |
| 2 | 0.5 | 1.75 | 1.5 | -0.5 | 2.0 | 1.0 |
| 3 | 1.75 | 0.5 | 2.0 | -0.5 | 1.5 | 1.0 |

Table 4: Order of sensitivities for each participant

| Stage | Time (s) |
|---|---|
| Control Stage | 30 |
| Test Stage | 60 |
| Aftermath Stage | 20 |
| Pause Stage | 60 |

Table 3: Time for each Stage in experiments

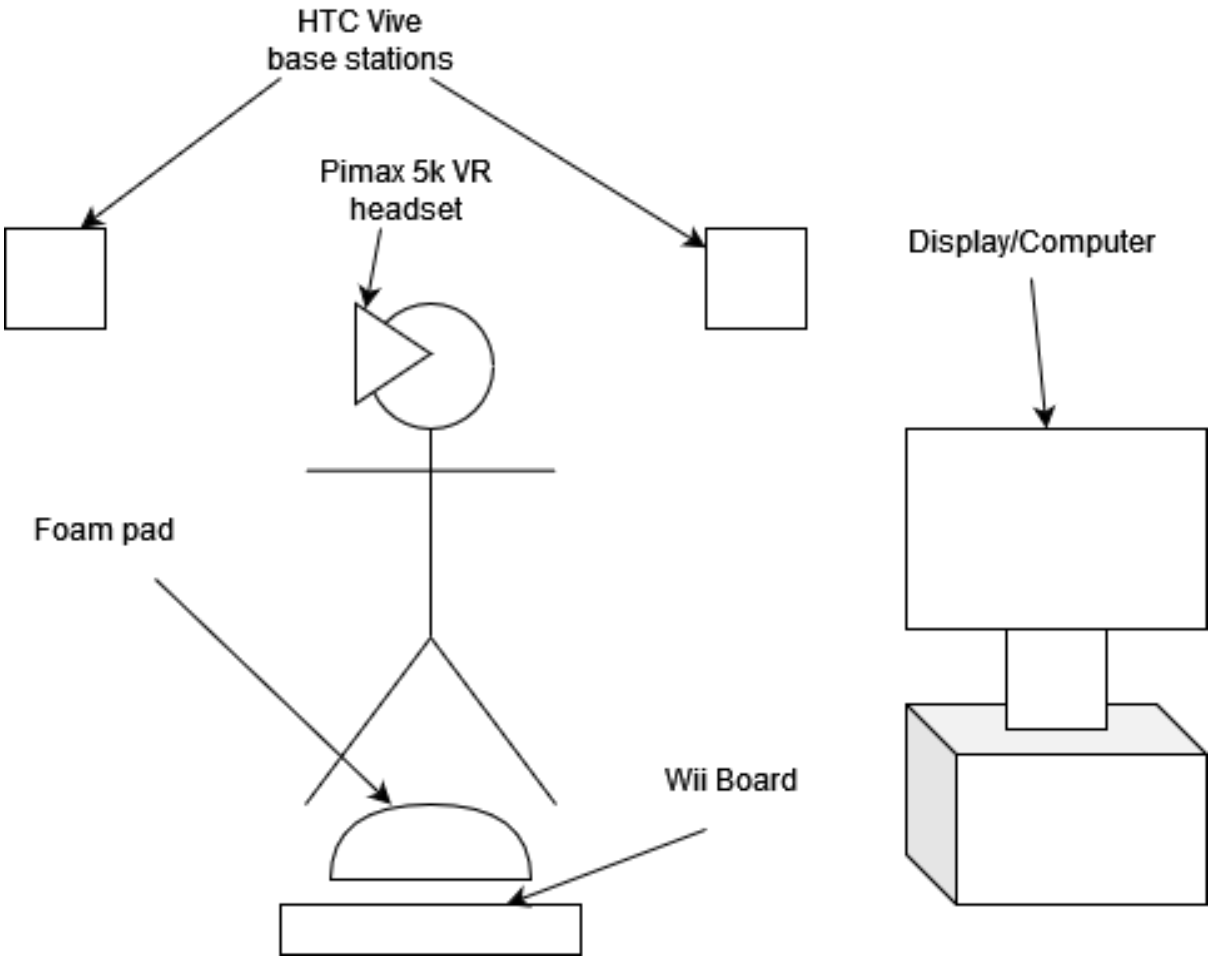


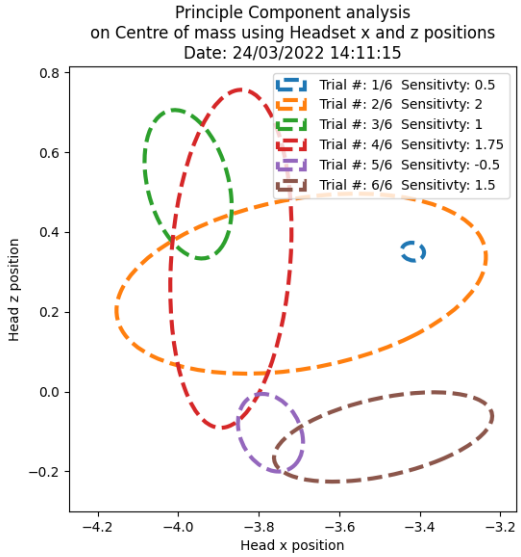

Figure 17: Set up for experiments

# 8 Results

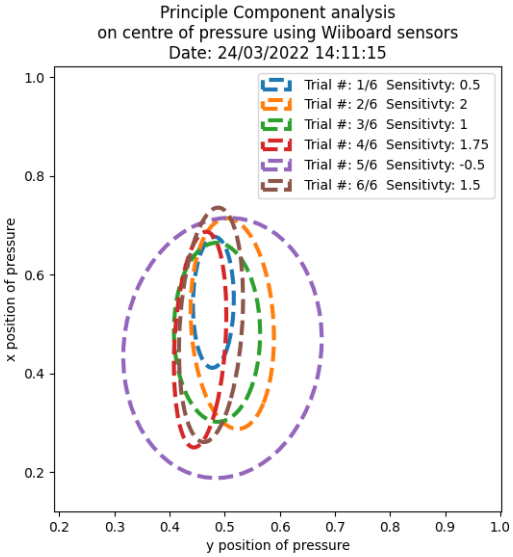
The confidence ellipses were generated by taking the head positions and centre of pressure positions, and applying principle component analysis to find the 2 axes that reduce the data with least loss of data. From there I use a 95% confidence margin for both axes (using the eigenvalues taken from principle component analysis as the variance of the data in each axis) to construct a 95% (2 standard deviations) confidence ellipsoid for each trial.

I then used the area of each ellipsoid in order to compare the change in sway for different participants.

## 8.1 Participant 1 results



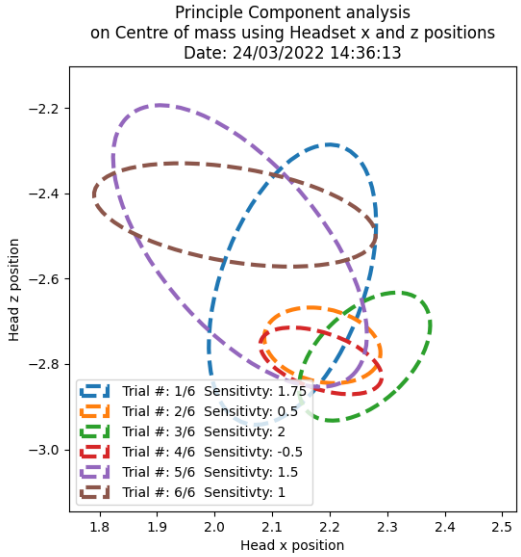

*Head positions 95% confidence ellipses*



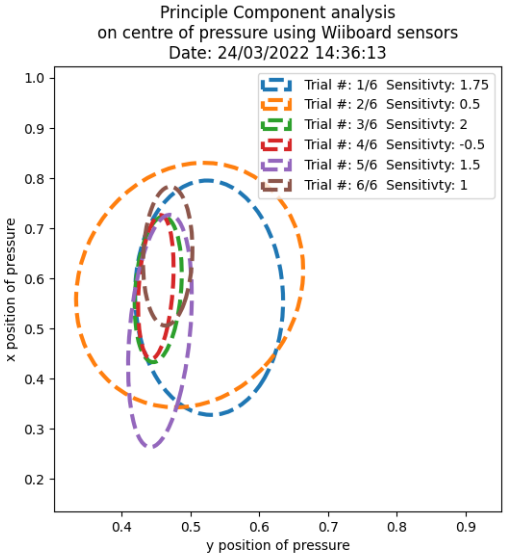

*Sensor positions 95% confidence ellipses*

Figure 18: Participant 1 Centre of mass (left) and centre of pressure (right) confidence (95%) ellipses

## 8.2 Participant 2 results



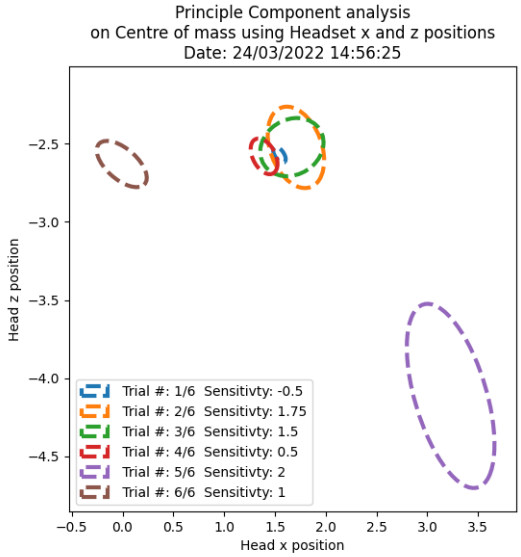

*Head positions 95% confidence ellipses*



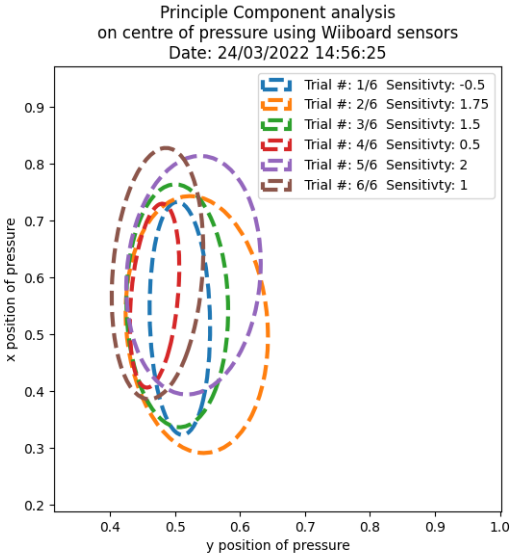

*Sensor positions 95% confidence ellipses*

Figure 19: Participant 2 Centre of mass (left) and centre of pressure (right) confidence (95%) ellipses

## 8.3 Participant 3 results



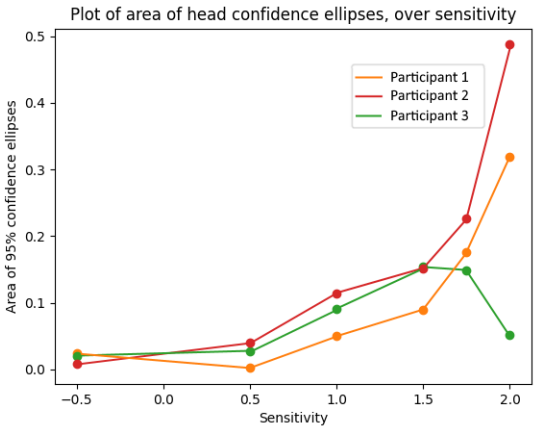

*Head positions 95% confidence ellipses*



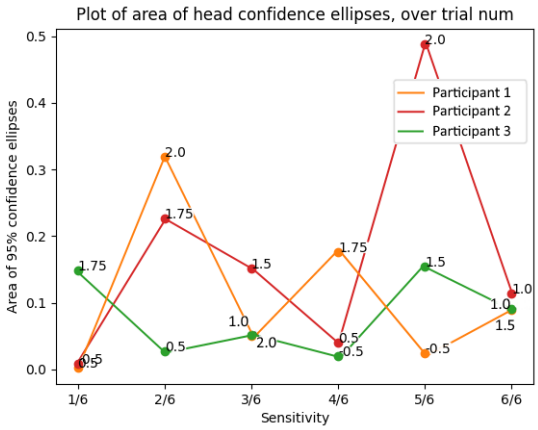

*Sensor positions 95% confidence ellipses*

Figure 20: Participant 3 Centre of mass (left) and centre of pressure (right) confidence (95%) ellipses

## 8.4 Accumulated results across participants

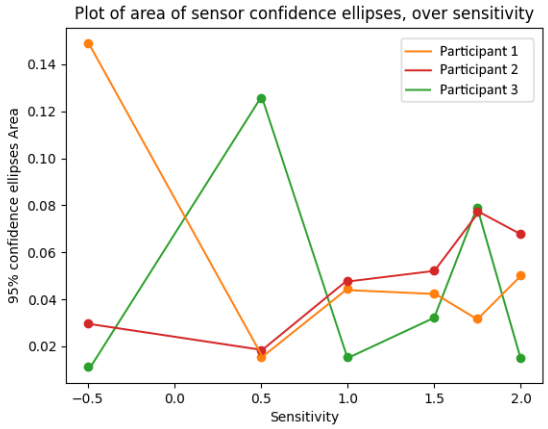



*Participants centre of mass total sway measured sorted by sensitivity*

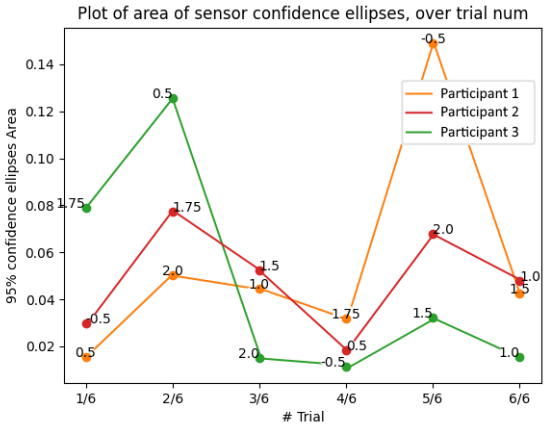



*Participants centre of mass total sway measured sorted by trial number*

Figure 21: Aggregate plot of each participant's centre of mass, sorted by sensitivity and trial number




*Participants centre of pressure total sway measured sorted by sensitivity*




*Participants centre of pressure total sway measured sorted by trial number*

Figure 22: Aggregate plot of each participant's centre of pressure, sorted by sensitivity and trial number

# 9 Discussion

The aim of these tests was to see if there is a correlation between distorting or augmenting visual sway, and human balance. Specifically, it is hypothesised that the more the visual sway is changed, the more unbalanced the person becomes. Furthermore, it was posited that should visual sway be too unrealistic or extreme, that human balance might not be affected.

## 9.1 Analysis of Raw Data

Looking at the raw collected data (Appendix 1), we can immediately see the effects of the sensitivities to postural sway and human balance. When comparing the central segment of the data, between the two red lines (signifying the Test Stage), we see a sharp change in the amount of variability in both head x and z positions, and the Wii Board sensor readings (examples : Figure 26, Figure 31, Figure 37). However this change is more distinct in the higher sensitivities than in the lower. For example, looking at Participant 3's sensitivity 2 Trial (Figure 36), there is a clear change particularly in head positions during the test stage compared to the control and aftermath stages. Comparing this to their sensitivity 0.5 Trial (Figure 38), the difference between Test and Control is much less discernible.

The raw data also shows an interesting detail on how the body reacts to a changing visual sway. At the beginning of the Aftermath stage, when the visual sway has made a direct return from augmented to usual 1:1, we can see Participants sometimes were unbalanced by this switch. For example the centre of pressure in Participant 2's 5th Trial (sensitivity 1.5, Figure 33), we see a sharp increase in the bottom sensors and sharp decrease in the top sensor readings. Comparing this to the "Control" stage we can deduce the Participant was off-balance, applying pressure on their heels in order to stabilise themselves. This post-Test off-balance does not seem to necessarily correlate with sensitivity, as same participant exhibited a similar reaction with the 0.5 sensitivity

trial (Figure 30) where this reaction appears in both the Wii Board readings and Headset readings.

## 9.2 Analysis of Ellipses

The ellipses are largely very oval shaped, and are mostly vertically oriented, particularly for the Wii Board readings. This was expected as the foam block the participants were asked to stand on invoked a strong back-and-forth sway as opposed to a left-to-right sway.

## 9.3 Analysis of Accumulated results

The accumulated results of sensitivities against head x and y sway shows a trend of positive exponential correlation. As the head position is the metric I am using to estimate centre of mass, this shows a correlation between physical postural sway augmenting in response to increased visual sway. However, interestingly, the centre of pressure does not have such an obvious trend. But, ignoring sensitivity value -0.5 as it is arguably a special case, we do see that both Participant 1 and 2 may have a trend. This is very inconclusive however and would require more participants to find out if this is the exception or the rule.

The second plot of postural sway against trial order does not have any obvious strong trends. However the centre of pressure counterpart appears to have a slight sinusoidal trend. To start with, each participant had lower scores for the first trial vs the second. This might be due to an expectation of what the next trial will be like based on the first, but then finding that the sensitivity is very different could have reduced their ability to balance. Each participant had a relatively large change in sensitivities (from 0.5 to 1.75, from -0.5 to 2, from 1.75 to 0.5, respectively) which supports this idea. Then it appears as though there is a "valley" in the 3rd and 4th trials. This might be due to participants getting more used to the environment and the unpredictability of the environment. The different participants sensitivities and change of sensitivities is very varied for these 2 trials. After this, all 3 participants exhibit a sharp increase in centre of pressure variance. Having received feedback from the participants, each one noted a sense of fatigue from the experiments, stating that this slightly hampered their ability to balance. This could explain the increase seen in this trial. The final trial for all 3 participants received a similar score in both postural and centre of pressure ellipsoid areas. This may suggest a bias where because the participants had done 5 trials prior, had began getting used to the environment (and the foam bar that was meant to apply some instability) and accommodating for the fatigue. It should also be noted however that the sensitivities they all were tested on were (by chance) very similar (1.0, 1.5, 1.0 respectively). Again, to gain any certainty in whether these are anomalous, or if the reasons suggested are the cause for this data, more tests with a larger sample would be necessary.

### 9.4 Affects of Negative Sensitivity

The negative sensitivity parameter was added as a brief investigation in the effects of what a decreased visual sway might have on postural sway, as well as a demonstration of how the application can be used to test situations outside of the original scope and intentions, without any additional programming. The results of this sensitivity are very interesting. Participant 1 has a very high disparity from centre of pressure and head positions for this sensitivity, for example. This implies that while the participant was able to stay balanced, a great deal of effort and pressure adjustment was needed to retain that balance. However this disparity is not seen in the other two participants. It is interesting to see that the results for head positions in this sensitivity are very similar and seemingly slightly less varied than that of the 0.5 sensitivity. This alludes to the idea that sensitivities between 0 and 1 (therefore visual sway that operates as at a fraction of usual sway) might increase balance, or tricks the body into applying less postural sway. A set of trials exploring the effects of sensitivity settings between 0 and 1 might yield interesting results for this concept.

# Part IV
# Conclusion and Evaluation

## 10 Conclusion of research

The results show an overall trend in the data to suggest that, by augmenting visual sway, postural sway was increased and human balance was decreased. The secondary part of the hypothesis, where heightened or unrealistic sway may result in less postural sway was not explicitly evidenced by the results. However, while in the process of creating the product, as many iterations of less realistic visual sway techniques were applied, it was clear that visual sway that poorly simulates what is really experienced when swaying does not provoke greater postural sway or imbalance.

## 11 Evaluation of research and experiments

Overall, the experiments and research were largely successful. The data gained from the experiments was sufficiently detailed and the efforts that were put in in removing some biases appeared to have paid off. However, in order to improve the experiments, for future research conducted on more people, there needs to be a few alterations to the experiments. The first is that, if the user actively puts too much effort in trying to balance, then they are able to do so with great success and therefore will hardly sway. This will result in them hardly experiencing the altered visual sway as their vision is not changing a great deal. A solution might be to ask the users to look for an item in the scene, and once found, the item will randomly teleport elsewhere for them to look for it again. This will ensure that there's no monotony in movement and the user will be forced to look around the scene. Another solution might be to apply a cognitive test which serves as

a slight distraction and consumption of cognitive ability- such as the Stroop Test. This will force the user to apply some thought to the test and therefore cannot concentrate on balancing perfectly. In my repository is a "StroopTest.cs" file which can easily be applied to the scene to create this functionality. The other key issue is the fatigue mentioned by the participants. The concern is that this might apply a bias to results. The solution for this might be to increase Pause Stage length.

Overall, these are valuable lessons to learn, and it is much better to have picked up these shortcomings by testing the experiments on a small focus group so that a larger sample is not potentially subject to these biases.

# 12 Evaluation of project

This project was overall a success. I set out to create a product that could be used for the research in this paper, while also be applied to future research in the area, with either little or no adaptation to the code. As a result, there is a great deal of potential experiments one can do with this application. For example, measuring the effects of sensitivities from 0 to 1 would be measuring the effects of a decreased visual sway. Measuring sensitivities of -1 or smaller would measure the effects of negative visual sway (the user forward, but the objects appear to be farther from the user).

By using a very simple and modular structure for the code-base, I allow future developers to easily modify the application in effective ways. For example, the Version 2 (Pendulum model of Sway) of the postural sway capture and augmentation is entirely interchangeable with the current model, and is compatible with all other code, by making use of a common base-class. Using this concept a developer could create an entirely new implementation of sway capture and switch out the current one without having to modify any other part of the code. Similarly, the method for creating different types of visual stimuli makes use of an enumerator, changeable from the Unity "inspector" menu, allowing someone to change between the various solutions without even having to open the C-Sharp file. The particle generation code also makes use of "delegate" function types. This allows for easy switching of particle generation without having to write the lines of code that will be the same for most implementations of particle generation (e.g. a for-loop of length : "number of particles"). There is also extensive documentation both in comment form within program files, and as readmes for both the Unity project and the C# Wii Board Socket. The readmes feature installation and troubleshooting guides, as well developer documentation on the code structure. The C# Wii Board Socket also features a detailed explanation of the syntax of packets sent over the socket, so that a future developer can program a receiver for a different project. The C# Wii Board Socket itself can be used for not only this project but truly any application that wants to make use of Wii Boards, in any programming language as long as there is a socket interface.

# Part V
# Appendix

## 13 Location of Git repository

A central git repository for this project is accessible via this link : https://git.cs.bham.ac.uk/lxa849/final year-project-2022. From there, the Unity components branch contains the relevant assets needed to create the Unity scene (and also contains the Python Scripts I used to create the included graphics). The Wii Board Socket branch contains all the source code for building the C# Socket - Wii Board application. Finally, in the releases section you can install the binary for both the unity project and the C# Socket application. Readmes are available on the root page and the branches for extra documentation and assistance.

## 14 Raw Results for all Participants

### 14.1 Participant 1 results

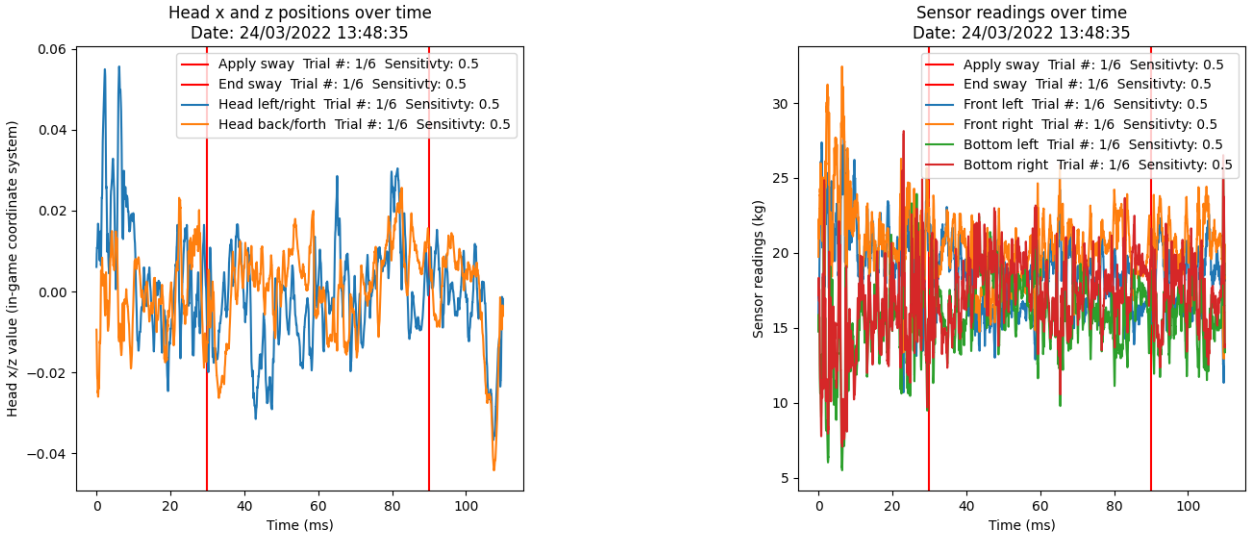



Figure 23: Participant 1 Trial 1

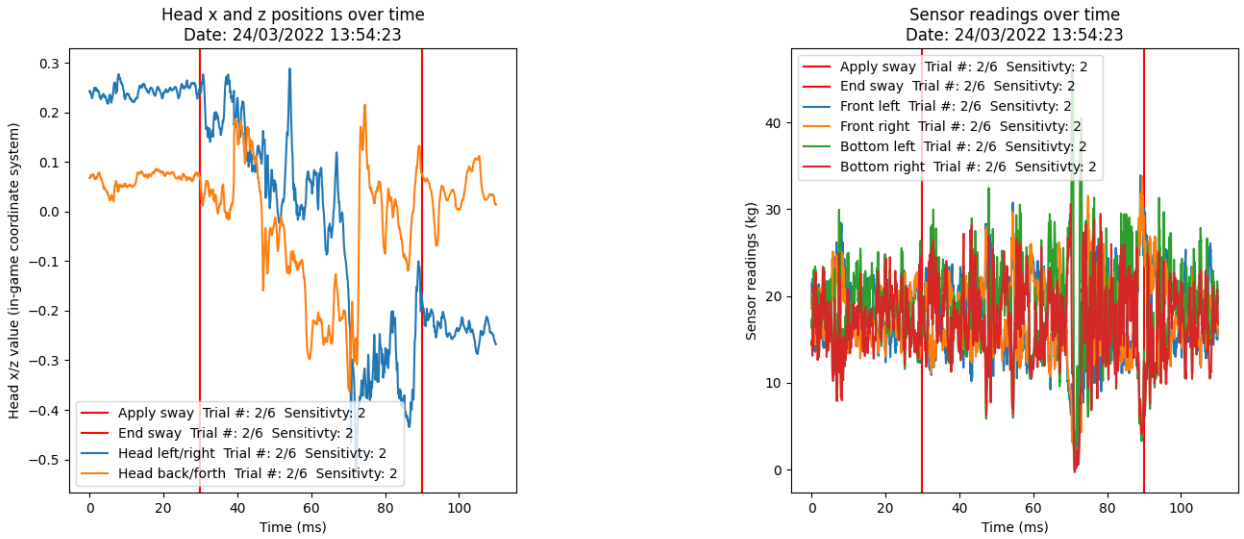



Figure 24: Participant 1 Trial 2

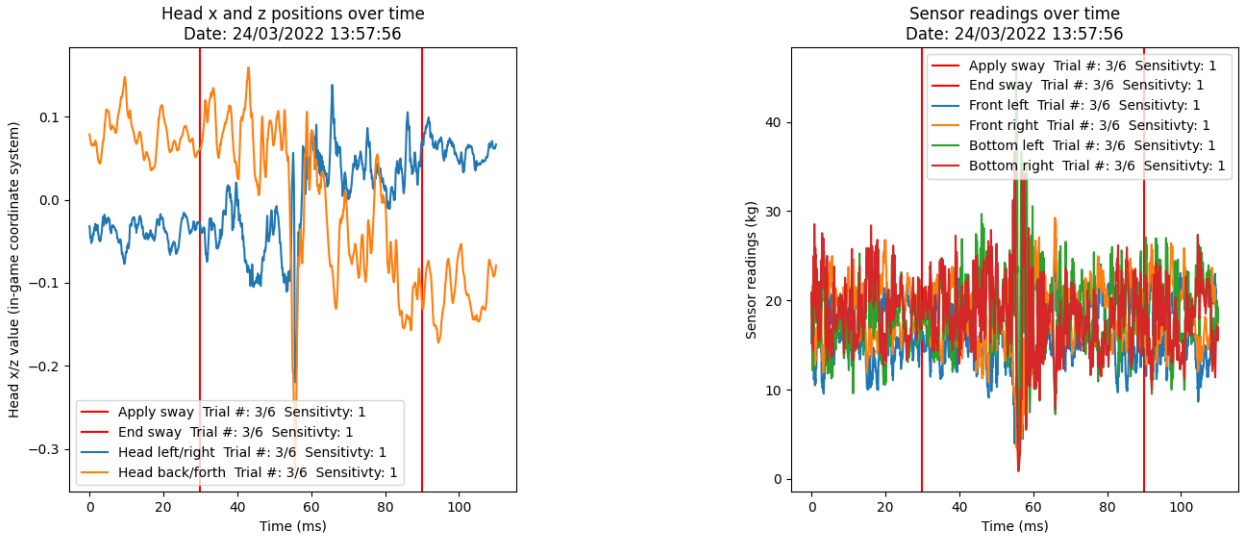



Figure 25: Participant 1 Trial 3

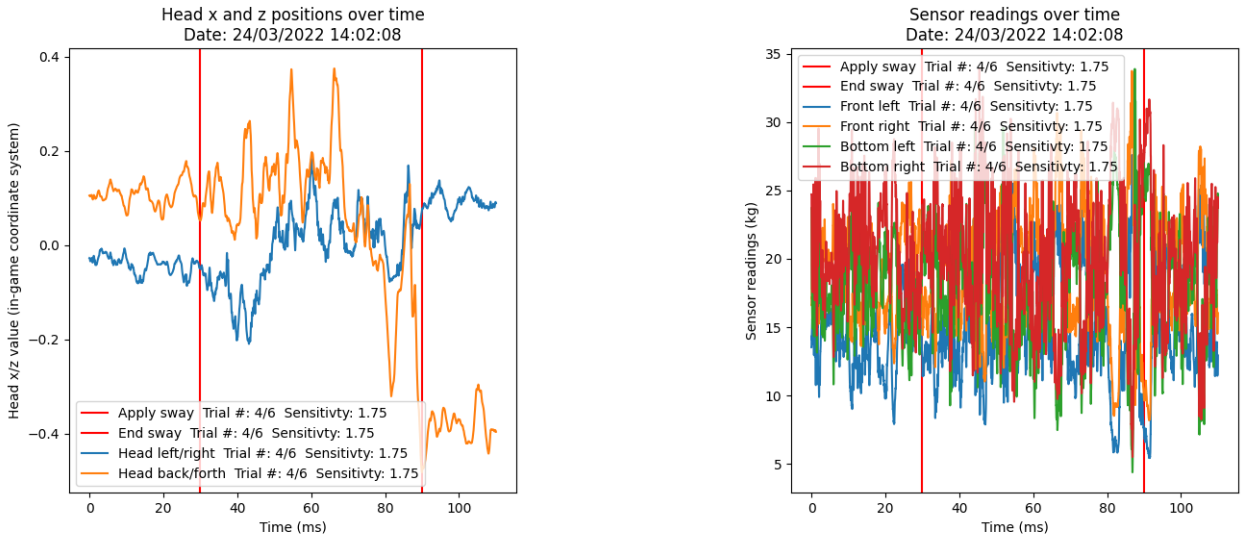



Figure 26: Participant 1 Trial 4

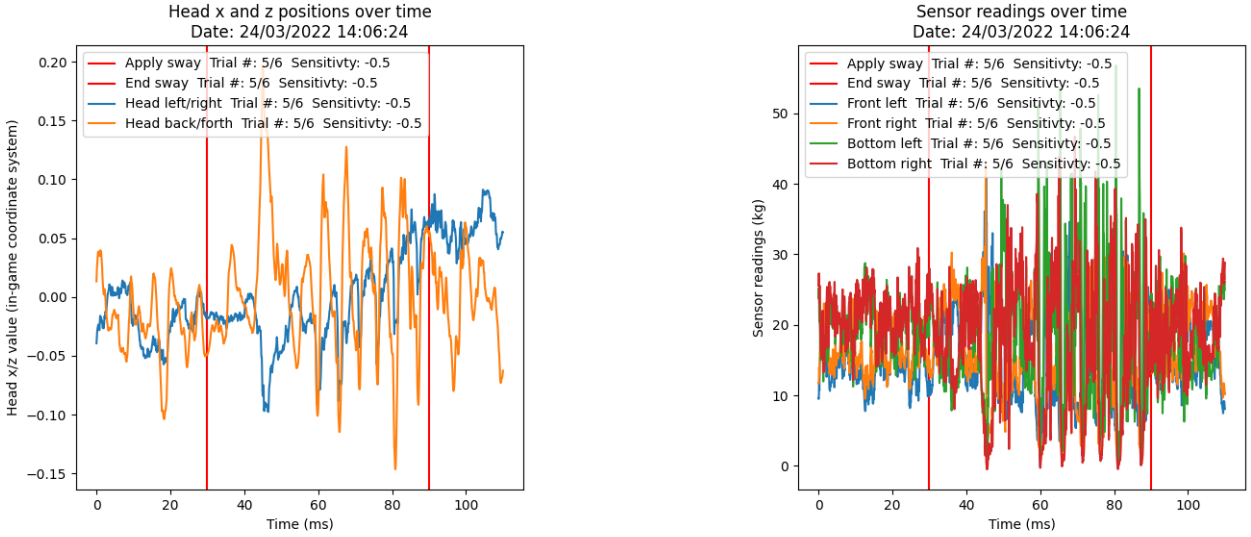



Figure 27: Participant 1 Trial 5

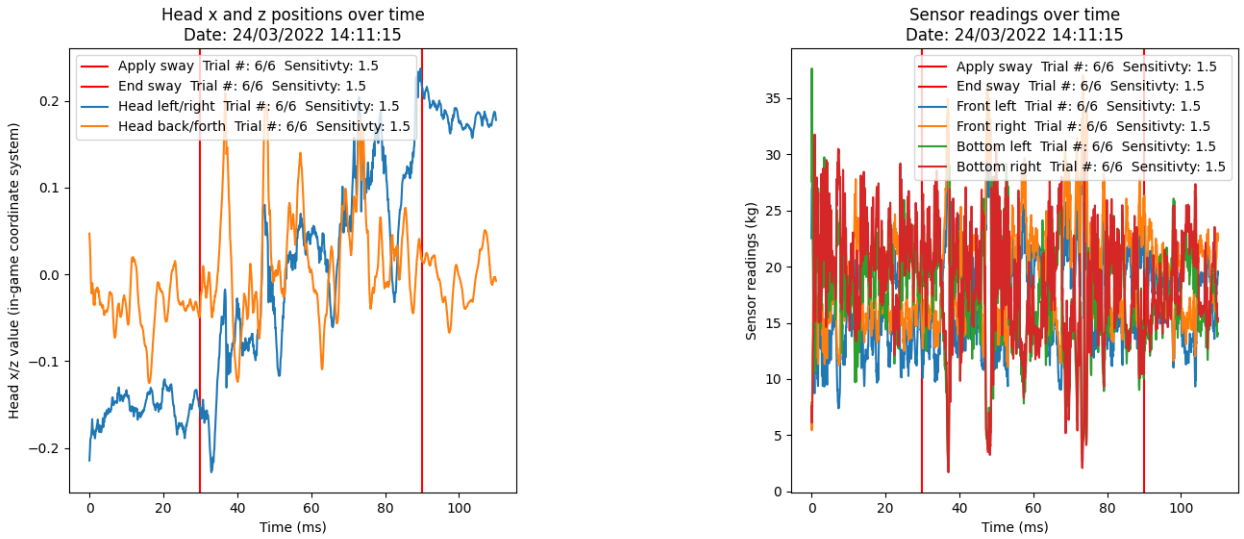



Figure 28: Participant 1 Trial 6

## 14.2 Participant 2 results

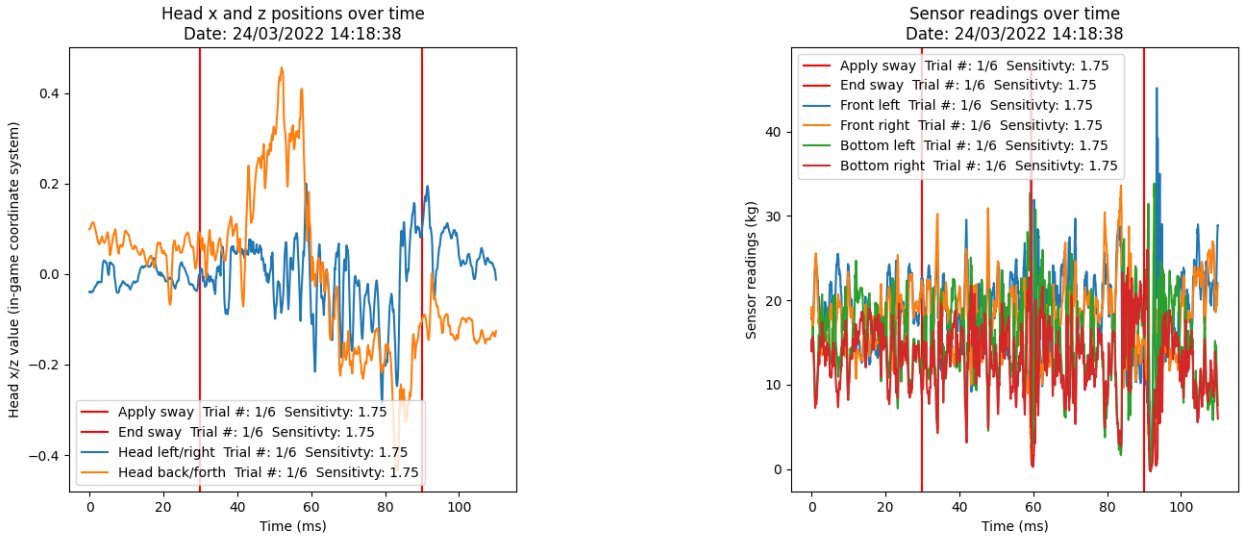



Figure 29: Participant 2 Trial 1

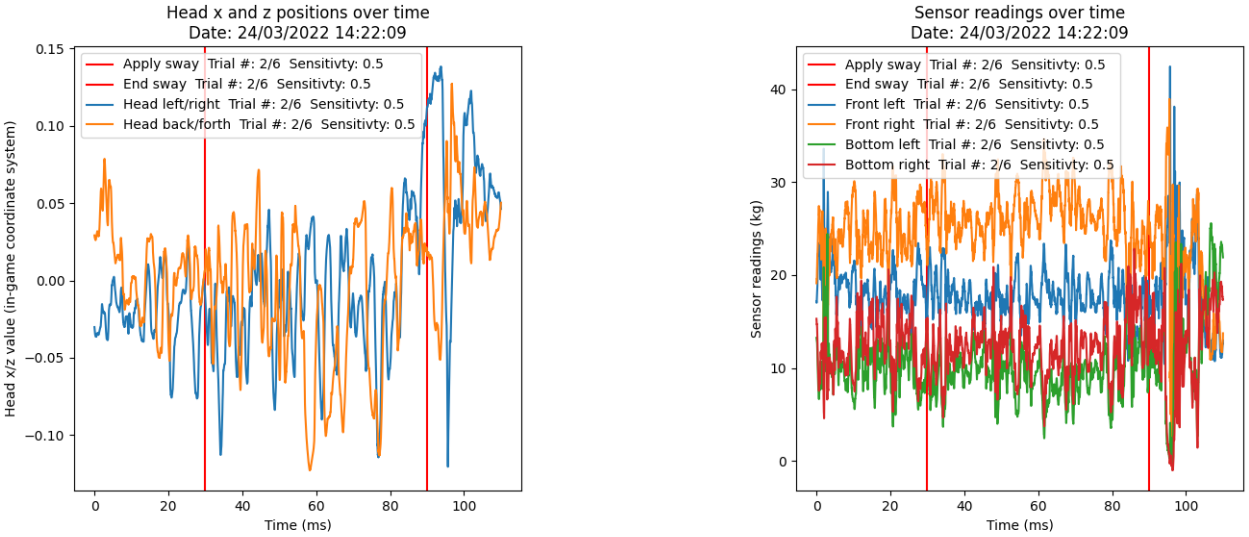



Figure 30: Participant 2 Trial 2

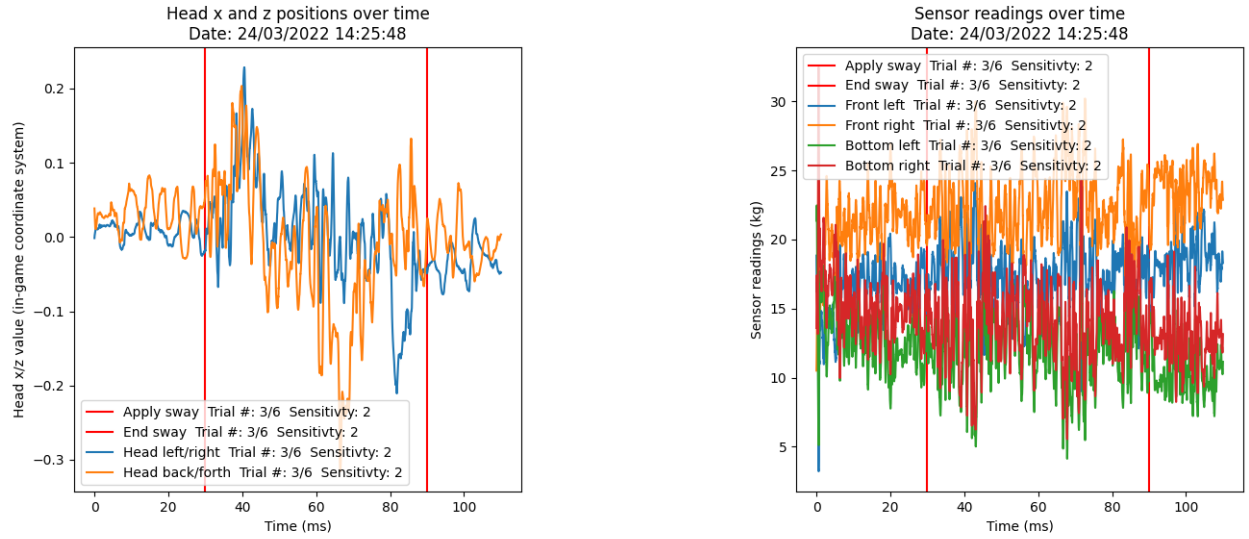



Figure 31: Participant 2 Trial 3

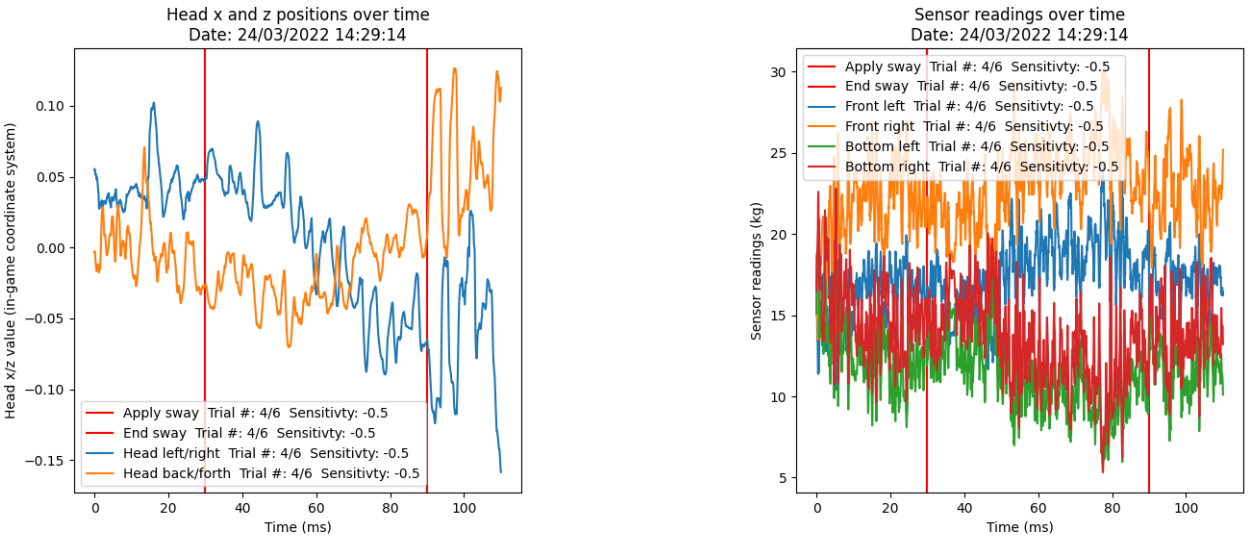



Figure 32: Participant 2 Trial 4

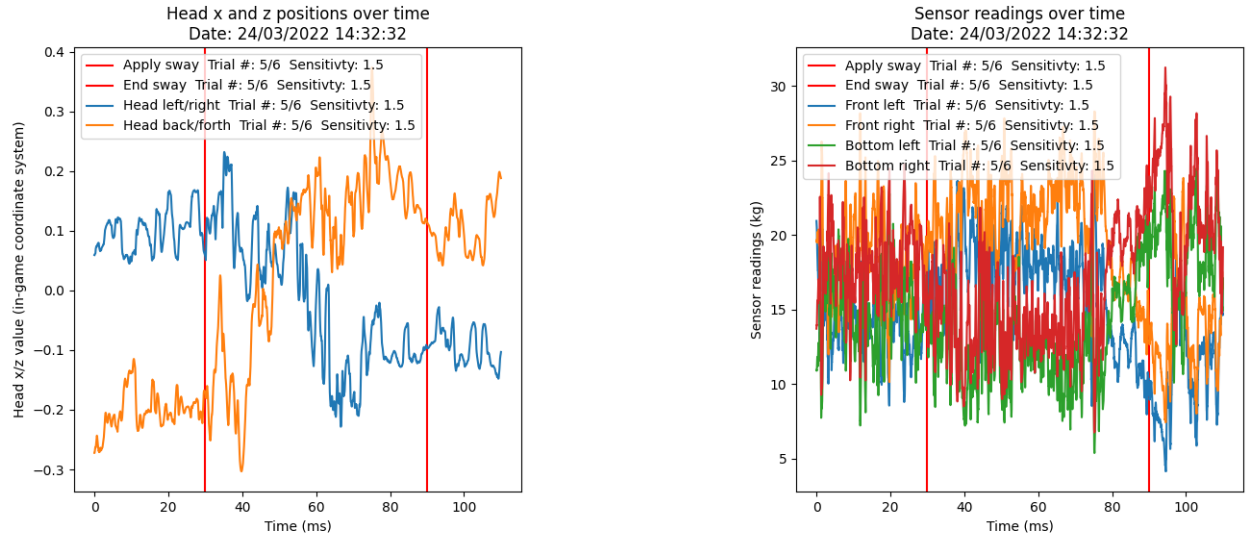



Figure 33: Participant 2 Trial 5

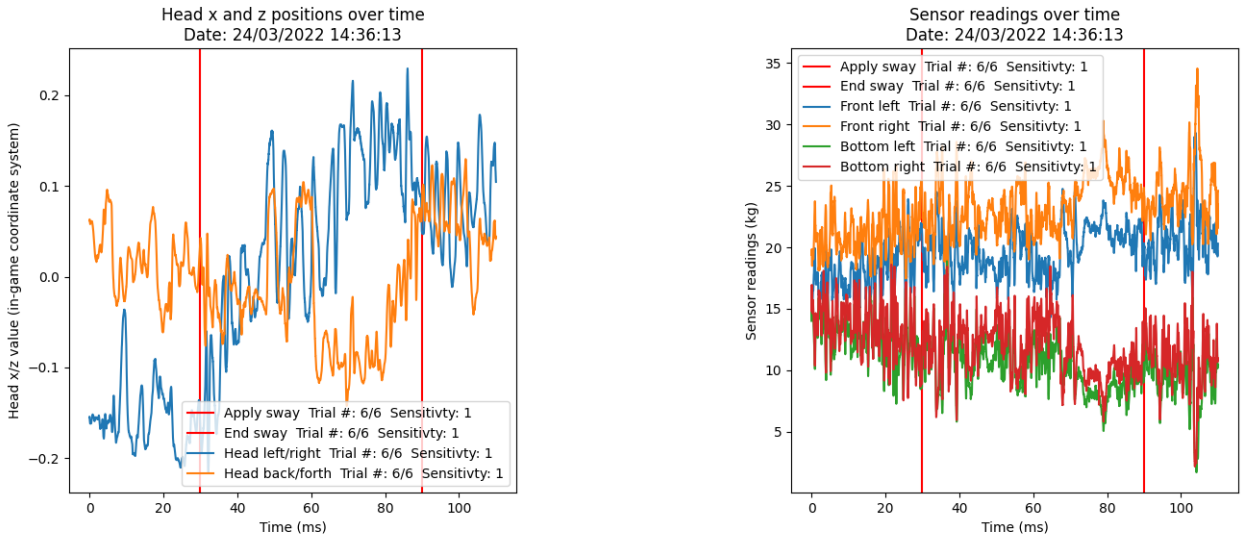



Figure 34: Participant 2 Trial 6

## 14.3 Participant 3 results

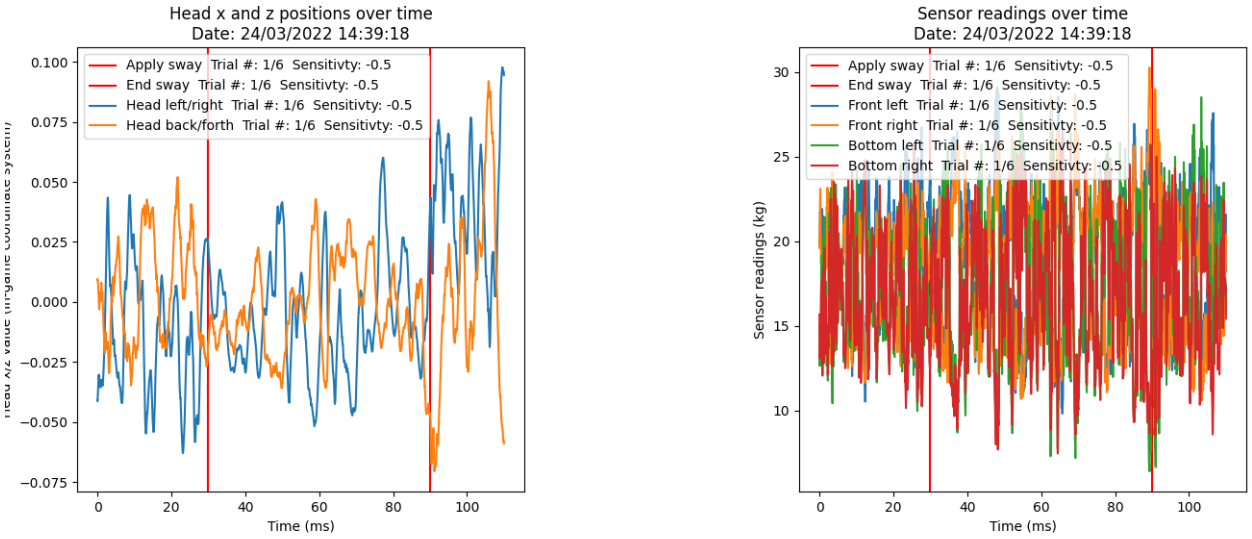



Figure 35: Participant 3 Trial 1

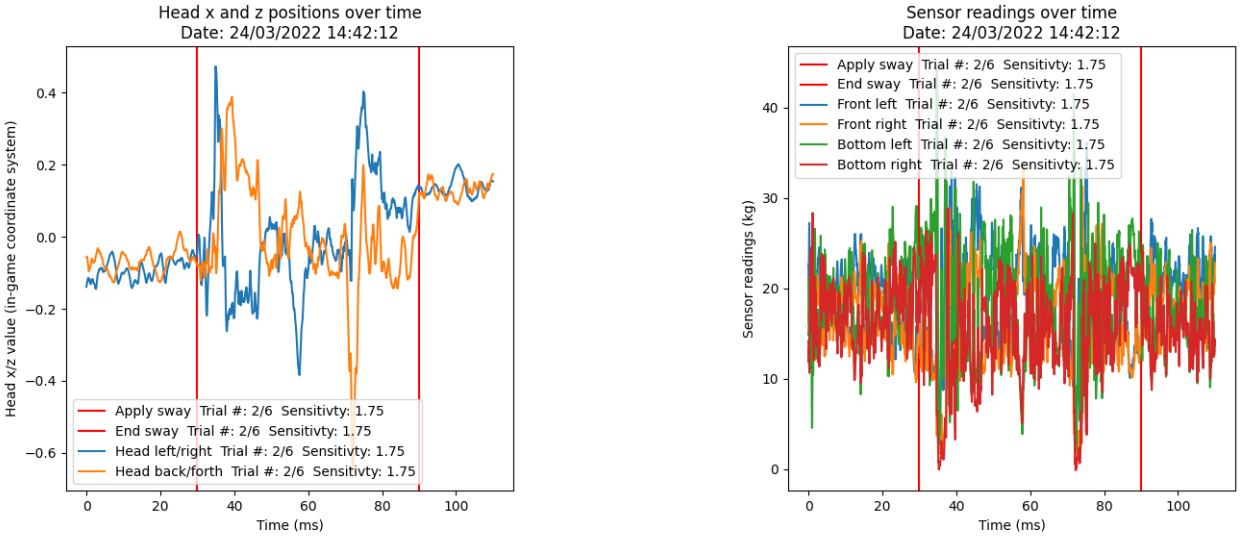



Figure 36: Participant 3 Trial 2

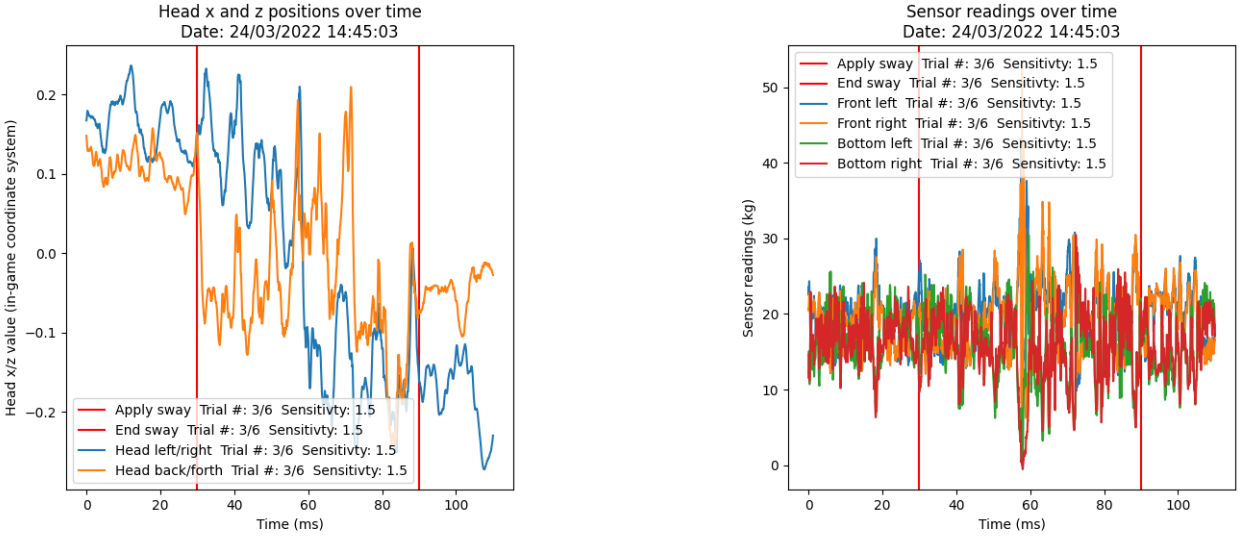



Figure 37: Participant 3 Trial 3

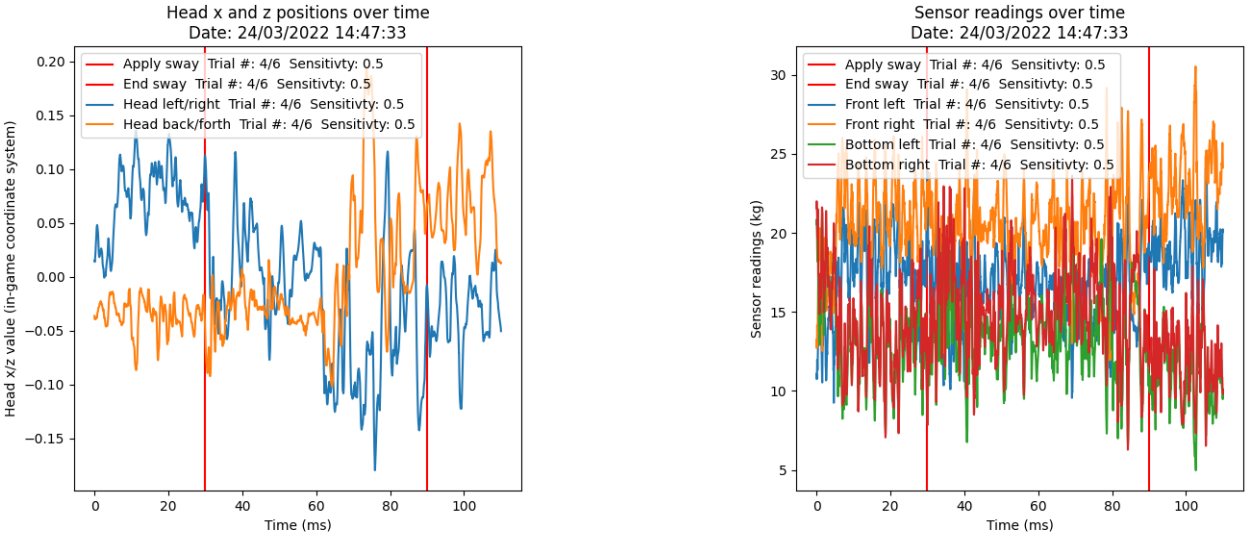



Figure 38: Participant 3 Trial 4

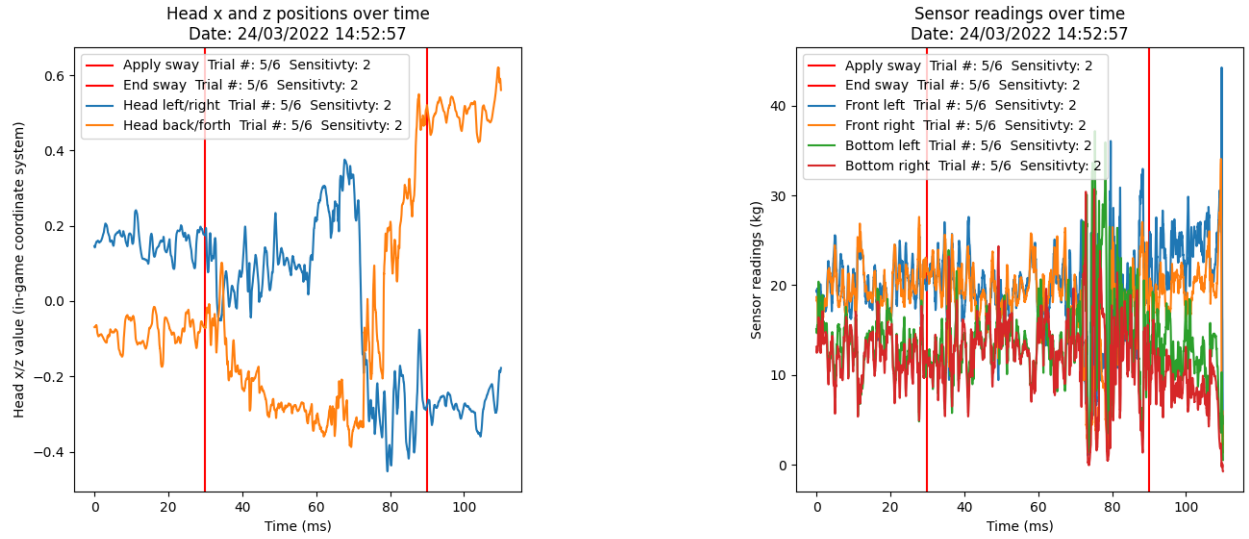



Figure 39: Participant 3 Trial 5

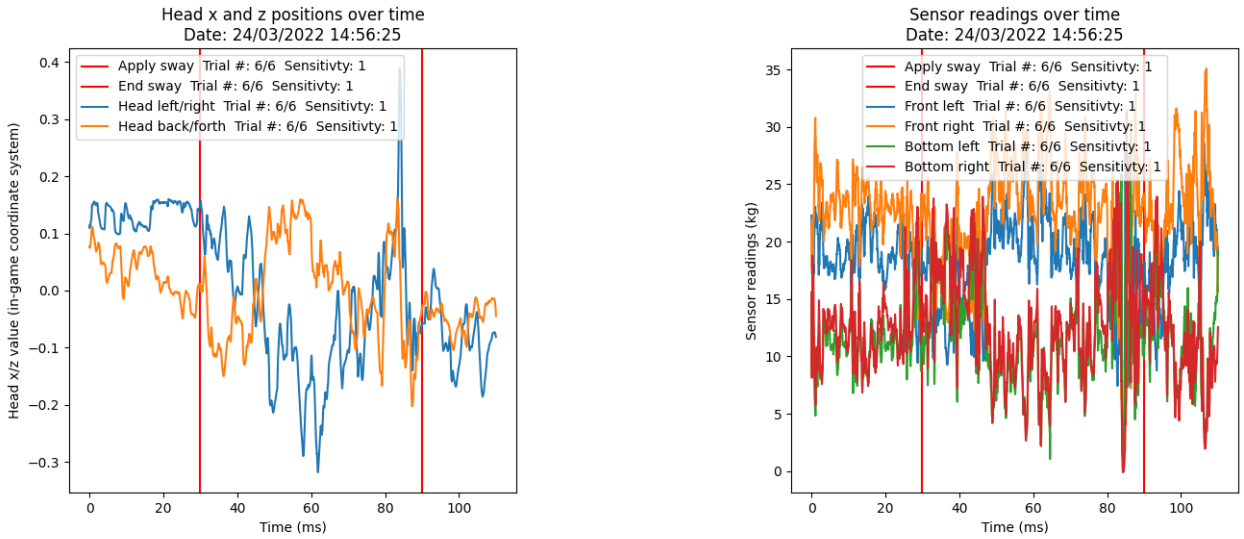



Figure 40: Participant 3 Trial 6

# References


[1] Gabriel Iuriciuc. "Developing a virtual reality environment for postural control experiments Author :" PhD thesis. 2019.

[2] Vivian Holten et al. "Interaction effects of visual stimulus speed and contrast on postural sway". In: *Experimental Brain Research* 234.1 (2016). ISSN: 14321106. DOI: 10.1007/s00221-015-4438-y.

[3] Wen Dien Chang et al. "Validity and reliability of wii fit balance board for the assessment of balance of healthy young adults and the elderly". In: *Journal of Physical Therapy Science* 25.10 (2013). ISSN: 09155287. DOI: 10.1589/jpts.25.1251.

[4] Ben Rohof et al. "The Nintendo® Wii Fit Balance Board can be used as a portable and low-cost posturography system with good agreement compared to established systems". In: *European journal of medical research* 25.1 (2020). ISSN: 2047783X. DOI: 10.1186/s40001-020-00445-y.

[5] Harrison L. Bartlett, Lena H. Ting, and Jeffrey T. Bingham. "Accuracy of force and center of pressure measures of the Wii Balance Board". In: *Gait and Posture* 39.1 (2014). ISSN: 09666362. DOI: 10.1016/j.gaitpost.2013.07.010.

[6] Julia M. Leach et al. "Validating and calibrating the Nintendo Wii balance board to derive reliable center of pressure measures". In: *Sensors (Switzerland)* 14.10 (2014). ISSN: 14248220. DOI: 10.3390/s141018244.

[7] Jill Guzman and Nadine Aktan. "Comparison of the Wii Balance Board and the BESS tool measuring postural stability in collegiate athletes". In: *Applied Nursing Research* 29 (2016). ISSN: 08971897. DOI: 10.1016/j.apnr.2015.04.008.

[8] Farbod N. Nezami et al. "Project Westdrive: Unity City With Self-Driving Cars and Pedestrians for Virtual Reality Studies". In: *Frontiers in ICT* 7 (2020). ISSN: 2297198X. DOI: 10.3389/fict.2020.00001.

[9] Jun Ho Huh. “Reliable user datagram protocol as a solution to latencies in network games”. In: *Electronics (Switzerland)* 7.11 (2018). ISSN: 20799292. DOI: 10.3390/electronics7110295.