

https://dspace.org/

Utilizing motion capture as an alternative input to promote presence in virtual environments

Gibson, James

2021
https://hdl.handle.net/10539/32155

Downloaded from a DSpace repository. https://dspace.org/

Utilizing Motion Capture as an Alternative Input to Promote Presence in Virtual Environments



UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

James Gibson

A dissertation submitted to the Faculty of Humanities, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of

Master of Arts in Digital Arts

Ethics Clearance Number: H19/11/16

Abstract

The aim of this project was to explore the feasibility of utilizing motion capture as an alternative control-scheme for games, with particular focus on leveraging it to promote presence in virtual reality environments. This was accomplished by utilizing a variety of classification techniques to classify the data provided by motion capture sensors, using those classifications as replacements for traditional handheld controllers. Multiple means of motion capture and classification were examined, with final implementation settling on the use of Xbox 360 Kinect sensors for capture and a combination of neural networks and dynamic thresholding algorithms for classification. None of the combinations of capture and classification techniques met the minimum framerate of 90 FPS required to be considered an outright success, although the high levels of gesture classification accuracy provided by many of these combinations proved the system to be feasible given the possibility of future work.

Keywords

Neural Networks, Virtual Reality, Motion Capture, Machine Learning, Kinect, Gesture Classification

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Arts at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other University.



(Signature of candidate)

_____30th_____day of _____March_____20____20_____at_____Randburg_____

Contents

1. Background 5

2. Core Theory 9

    2.1. Virtual Reality 9

    2.2. Motion Capture 11

    2.3. Kinect 13

    2.4. Neural Networks 15

3. Methodology of Practical Work 21

4. System Implementation 26

    4.1. Threshold Stances 27

    4.2. Neural Network-Based Stance Classification 30

    4.3. Classification Implementation in Unity 32

    4.4. Implementing Bending Effects 34

    4.5. Utilizing Multiple Kinect Sensors 36

    4.6. Player Avatar and Body-Matching 38

    4.7. Moving to Virtual Reality 41

    4.8. Revisiting Thresholding 42

    4.9. Networking and Multiplayer 44

    4.10. Stance Enrolment and Neural Network Training 45

5. Testing the System 47

6. Results 48

6.1. Solo Networks 48

6.2. Bracket Networks 49

6.3. Master Networks 51

6.4. Threshold Classification 51

6.5. Kinect Setups 52

6.6. Framerate 53

6.7. Stance Similarity 54

6.8. Classification Precision 54

6.9. Practical Application of Results 55

7. Future Work 56

8. Conclusions 59

9. Results Tables 61

10. References 68

11. List of Figures 71

12. List of Tables 72

1. Background

Virtual reality (VR) systems such as the HTC Vive and Oculus Rift have been available for purchase since 2016, with their earlier, more development-oriented versions having first been showcased and distributed as early as 2013. Since their respective consumer releases, these systems have become increasingly popular. The steady growth in VR use can be attributed to a number of factors, among them being technological improvements, an ever-widening range of options to choose from, and adoption by businesses for training purposes (Dujmovic, 2019). Many of the more popular VR games currently available to consumers are controlled through a set of simple interaction mechanisms, namely button presses and arm movements, all of which are tracked by the handheld controllers. While these interactions are leveraged well to drive gameplay mechanics, there is much room for more complex interactions that could greatly expand the possibilities in these games.



Figure 1 – Promotional screenshot of Beat Saber from its Steam store page. 21 May 2019. Beat Games. https://cdn.cloudflare.steamstatic.com/steam/apps/620980/ss_1881ae4f153faf0d1ccecca60fbdac5b43ad57eb.600x338.jpg?t=1603721346. Accessed 10 October 2019.

A prime example of this type of game is *Beat Saber* (Beat Games 2018), which prompts the player to move their controllers in order to slice incoming blocks in a variety of directions based on the beat of a selected a piece of music. While the game has long been one of the best-selling VR titles

on Steam, the experience it provides is limited to small windows of play, rather than providing the player with a deeper, more immersive experience. While players can experience a deep sense of immersion from playing games such as this, they do not offer an environment that promotes suspension of disbelief.

On the other end of the scale, a game like *The Forest* (Endnight Games Ltd 2018) provides the player with an open world in which they are required to hunt, explore and harvest resources in order to survive. Actions that have previously been triggered using button presses and mouse or analogue stick movement, such as cutting down trees with an axe or aiming and throwing a spear, are now more nuanced, requiring arm swings that mimic the actual movement. While this type of interaction is precisely the same as the one utilized in Beat Saber, the world in which the game takes place and the agency that these interactions afford the player greatly promote suspension of disbelief and a sense of presence.



Figure 2 – Promotional screenshot of The Forest from its Steam store page. Endnight Games Ltd. 30 April 2018. https://cdn.cloudflare.steamstatic.com/steam/apps/242760/ss_d03a261fecab226a0ecac5746225c2a50d65c670.600x338.jpg?t=1590522045. Accessed 10 October 2019.

Current mainstream VR products use a headset and two controllers as the source of input for the player, using various mechanical and optical sensors to track the positions and rotations of these

3 objects in space. The size of this play-space can be expanded by introducing more optical sensors to cover a larger effective area. These tools and sensors work together to immerse the players in their games in a way that traditional screens cannot compete with, but break the illusion when the players realize that they still need to use a touchpad or analogue stick to move their characters or are unable to see their hands, arms and other parts of their bodies while looking around.

Mel Slater and Sylvia Wilbur proposed the following in their paper titled *A Framework for Immersive Virtual Environments (FIVE)*:

> The degree of immersion can be objectively assessed as the characteristics of a technology, and has dimensions such as the extent to which a display system can deliver an inclusive, extensive, surrounding, and vivid illusion of virtual environment to a participant. Other dimensions of immersion are concerned with the extent of body matching, and the extent to which there is a self-contained plot in which the participant can act and in which there is an autonomous response. (Slater and Wilbur, 1997, p. 603)

While the project reported here does not include anything akin to a narrative plot, the rest of the points outlined in this statement provide an idea that immersion can be quantified through a number of different metrics. Possibly the most important point mentioned here is the idea of "body-matching", something that is currently limited to hand and head position on current VR systems. As the intention of this project is primarily a proof of concept, the most important thing is to create a controlled environment in which accurate testing can be done. Narrative has little place in such an environment as its addition would most likely hinder progress on other parts of the system or skew testing data.

This project aims to take body-matching a step further by using motion capture sensors to replicate full-body gestures performed by the player in the virtual environment, allowing them freedom to walk, jump and dodge within the capture space. Once this is possible, the analogue sticks and track pads of traditional VR controllers can be done away with as movement controls. The same can be said for the positional tracking provided by the controllers, leaving the buttons and triggers as the only input sources that need to be replaced. This is where elements of machine learning come into play.

Machine learning is a particular subset of artificial intelligence that is used to examine large sets of training data in order to find patterns and trends, forming links between that data's input and output values. Once these systems have been taught recognize the necessary patterns, they provide informed outputs based on any input they are supplied. Leveraging these techniques will provide the system with accurate and customizable classification tools, provided they are trained correctly. In particular, neural networks make for very efficient classifiers when tasked with interpreting numerical data such as vector-based coordinates and distances, all of which can either be read directly from motion capture sensors or extracted from the raw data they provide.

If a neural network can be used to accurately classify individual gestures in real-time, these gestures could act as a replacement for the buttons and triggers on VR remotes, doing away with the last reason they would be needed. The system would be able to provide a virtual environment in which all inputs are derived solely from the movement of the user's body. A system capable of achieving this requires three main components; a method of capturing data from the user's body, the ability to use that data to create an accurate digital representation of the user in the virtual environment, and a classifier capable of using that data to translate the user's gestures into actions triggered in the environment. A system that makes use of all of these things should greatly enhance the user's sense of presence.

Slater and Wilbur define presence as follows:

> "Presence is a state of consciousness that may be concomitant with immersion, and is related to a sense of being in a place. Presence governs aspects of autonomie responses and higher-level behaviors of a participant in a VE (Virtual Environment)." (Slater and Wilbur, 1997, p. 603)

While presence is a sense experienced differently by individuals and is thus too subjective to be quantified, Slater and Wilbur (1997) believe that it is strongly linked to the various immersion metrics mentioned previously. In order to create an environment that enables and promotes a feeling of presence for people who experience it many methods of motion capture, body-mapping and gesture classification will need to be examined in order to find the best option in terms of each of the immersion metrics.

2. Core Theory

The concepts of immersion and presence in virtual environments as proposed by Slater and Wilbur (1997) have already been discussed. Despite their paper having been published in 1997, the ideas discussed are still very relevant and Slater was awarded the "Virtual/Augmented Reality Career Award" in 2005 (IEEE VGTC).

In Janet Murray's 1997 book, *Hamlet on the Holodeck: The Future of Narrative in Cyberspace*, she touches on a large number of interesting topics that remain highly relevant 20 years later. She mentions 'the Four Essential Properties of Digital Environments', stating that these environments need to be procedural, participatory, spatial and encyclopedic. She goes on to mention that the procedural and participatory properties are what help define how interactive the environment is, while the remaining two make the environment seem more explorable and extensive (Murray, 1997).

This project is intentionally disregarding the encyclopedic property as it does not fit within the scope of what the system is trying to achieve, namely, an enhanced sense of presence brought about by allowing users to physically interact with the digital world as naturally as they would in the real world. The remaining three properties are all core to the project in their own respective ways, governing its rule-based nature regarding particular martial arts movesets, the level of interaction afforded to the user, and the clarity and fluidity of how the play space can be navigated.

Murray (1997) stated that "When we are immersed in a consistent environment we are motivated to initiate actions that lead to the feeling of agency, which in turn deepens our sense of immersion. This phenomenon can be thought of as the Active Creation of Belief". The system should thus accurately replicate all of the physical actions performed by the user, allowing them to experience just such a "creation of belief".

2.1. Virtual Reality

The goal of this project is not to attempt to re-invent the wheel, but rather to use existing

technologies in new combinations to achieve accurate control of a VR system using full body motion capture. The first requirement is to touch on the different VR devices available to consumers, most importantly the HTC Vive as it is the primary device that the system will use for testing. This project takes the form of a feasibility study, and thus is directed more towards the low end of the spectrum in terms of devices to be utilized. As most dedicated HMDs (head mounted displays) sport the same 2160 x 1200 resolution and 90Hz refresh rate, the quality of visual fluidity needed for this project is easily accessible (Lamkin, 2018). These HMDs are built to strap onto the user's head and house the screen or screens that visualize the virtual environment, as well as any audio output devices or ports to plug such devices into.

The unusual resolution width for these HMDs is needed as they are catering to the wearer's eyes individually, thus effectively providing each eye with a 1200 x 1080 screen. The high number of pixels increases image sharpness and in so doing also increases immersiveness (Slater and Wilbur, 1997, p. 603). The 90Hz refresh rate on these screens is a necessity when it comes to avoiding flicker. This is especially important considering the wide field of view provided by HMDs and the human eye's tendency to be more sensitive to flickering in the periphery (Perz, 2018).



Figure 3 - Oculus Rift (left) and HTC Vive (right) HMDs from a comparative review on Tech Advisor. Painter, Lewis. 27 May 2016. https://www.techadvisor.co.uk/cmsdata/reviews/3641088/htc-vive-oculus-rift-main_thumb800.jpg. Accessed 15 July 2018.

2.2. Motion Capture

Many methods of animation have been developed through the years, ranging from the original hand-drawn animations and rotoscoping of early cartoons to the much more technologically advanced means provided by the motion capture studios that are so popular today. While almost no animations are completed without at least a touch-up from an animator, motion capture provides the quickest path from an idea for an animation to its digital implementation.

Before motion capture's rise in popularity and use, methods such as keyframe animation were used to simplify the animation process, allowing animators to define only the most important frames in each movement. The system can then interpolate between these frames, yielding a fluid animation that passes through all of them in a specified order.

Motion capture can be achieved by a variety of means that can be split up into two major categories; optical and non-optical. Non-optical capture is performed by putting the subject in a suit housing various sensors that report on the locations of various body parts within the capture space.

There are three types of sensors that can be used in such a suit, namely inertial, magnetic and mechanical. In the case of inertial sensors, small devices such as accelerometers and gyroscopes are used to provide angular and movement data, which can be fed into various kinematic algorithms to provide accurate positional data (Roetenberg et al., 2013; Schepers et al., 2018).

Magnetic capture also utilizes multiple small sensors located throughout a suit that provide positional and rotational data by determining their locations within a controlled magnetic field (Motion Capture: Magnetic Systems, 1995). A major issue that this method has is its reliance on cabling between the magnetic sensors and their central control unit, drastically reducing the subject's freedom of movement (Lonkar, 2018).

The last and most rigid form of non-optical capture is mechanical. The suit used in mechanical capture can be better described as an exoskeleton, a series of metal rods and joints attached to the subject's back. This rigid structure makes fluid motion much more difficult to achieve than the previous two suits, and is thus much more suited to key framing than full capture. The exoskeleton

also has no spatial awareness, so actions such as jumping cannot be accurately recorded (Lonkar, 2018).



Figure 4 - Xsens motion tracking suit and its digital mappings from a web article on Engadget. Haradawar, Devindra. 17 June 2017. https://www.engadget.com/2017-06-17-xsens-motion-capture.html. Accessed 15 July 2018.

Optical motion capture makes use of cameras as its name implies, visually capturing data from the subject rather than utilizing small sensors on a suit. This does not mean that no suit is required, as in the case of marker-based capture. Active markers come in the form of LEDs located throughout a suit, each lighting up one at a time in quick succession during the capture process. The ongoing positions of these LEDs are then triangulated using a series of cameras placed around the outer edges of the capture space. This method is often referred to as Pulse LED capture (Lonkar, 2018, Optical Motion Capture Systems, 2018).

Passive markers function in very much the same way as their active counterparts, with the major difference being the source of the light being captured. Passive marker suits provide the subject with a number of reflective patches located throughout their body and, as they move, light from sources around the capture space is reflected off of these patches and captured by the camera array (Guerra-Filho, 2005). This method of capture provides the most freedom of movement of any of the suit-based methods (Lonkar, 2018).

The final form of optical motion capture is known as markerless capture, and is performed without any need for a suit or other wearables. This makes it by far the most accessible form of motion capture, requiring nothing more than one or more cameras and the appropriate software. Companies such as *wrnchAI* are using artificial intelligence to extract positional data from ordinary video footage taken from a single camera, opening up a whole new world of possibilities and expanding the bounds of what can be achieved with motion capture systems with their 'pose estimation' concept (wrnchAI, 2018).

While the other forms of motion capture use precisely placed sensors or markers to map out the subject's movements, markerless capture relies heavily on its software aspect, requiring the various algorithms at its disposal to clean up visual noise, extracting the subject from the background and estimating the positions of key points such as bones and joints based on the visual data extracted. This is heavily limiting especially considering how detrimental occlusions are in optical capture (Berger et al., 2011, Markerless Motion Capture using multiple Color-Depth Sensors, 2011), but provides a very good baseline for the project. Occlusions are the general term given to situations where an object being optically captured is blocked (occluded) from sight by another object.

In theory, if what is arguably the least accurate form of motion capture can be used to control a virtual system, then the results can only improve as more accurate technology is introduced. The accessibility of this form of capture also makes it very appealing as it has the greatest potential to affect a larger number of people in practice.

2.3. Kinect

The Xbox Kinect sensor is the most readily available commercial motion capture device for this project. The first version of the sensor was released in late 2010 with the intention of expanding the Xbox consumer pool (Pham, 2009). At the time, the company was probably unaware as to how much its device would be used as a computer vision tool. Computer vision is a term that primarily describes systems that extract data from images, or series of images in the case of videos, and interpret and extrapolate on that data to provide more high-level information, such

as what types of objects are depicted in an image. Less than half a year after the device's release, Microsoft released the Kinect SDK, providing researchers and enthusiasts with the tools they needed to develop their own applications for the sensor (Orland, 2011).



Figure 5 - First generation Kinect sensor from the Amazon UK store. https://images-na.ssl-images-amazon.com/images/I/31pypkkCMZL._AC_.jpg. Accessed 15 July 2018

The Kinect sensor contains three key devices that work together to provide motion capture data, namely the infrared emitter, depth camera and colour camera. The infrared emitter projects infrared light into the capture space, allowing the other cameras to capture up to 30 frames per second in any light conditions (Totilo, 2009). The time taken for the infrared light to be captured by the camera after being emitted and bouncing off of an object is known as 'flight time'. This is calculated per pixel to determine the depth at each point in the captured frame, building a three-dimensional representation of the capture space.

With the capture subject placed a reasonable distance away from any obstructions, the Kinect can fairly easily extract the subject's positional data and reconstruct it digitally as a 20-joint skeleton in three-dimensional space (Cong and Winter). These structures of positional data are readily available as an array through the Kinect API, allowing them to be used easily for any number of applications. In 2013, an upgraded version of the Kinect was released for the Xbox One console, sporting a large number of improvements over its predecessor. The improvements made to this sensor made capture more accurate while simultaneously expanding the size of the capture space.

While the Kinect sensor provides a seemingly easy means of motion capture, it still suffers from the same issue as all forms of optical capture - occlusions. For instance, if the subject points one arm directly at the sensor, the only thing visible to that sensor is the subject's hand. All data regarding the position of the unseen shoulder, elbow, and wrist joints are estimated based on the positions of the seen hand joint. These estimations are seldom accurate, and often fail to estimate positions altogether.

Fortunately, the usual manner in which the occlusion problem is solved can be applied to the Kinect sensor as well; namely, incorporating more sensors. Using multiple Kinects facing the subject from different angles means that when an occlusion occurs for one sensor, one of the other sensors can be used to fill in the missing data (Berger et al., 2011, Markerless Motion Capture using multiple Color-Depth Sensors, 2011).

Overlaying the positional data from multiple sensors requires more calibration than using a single sensor, as the distances between sensors and their various angles of rotation need to be taken into account before an accurate mean capture can be achieved. With the exception of instances of poor calibration, the mean capture provided by an array of sensors will always be more accurate than the data provided by a single sensor.

The purpose of motion capture within this project lies somewhat outside of the norm, being directed at live replication rather than capture and replay after the fact. With all the data provided by one or more of these sensors, it is imperative that there are tools available to analyze this data in real-time. Many recent advances in this sphere of technology now make this achievable.

2.4. Neural Networks

In recent decades, the world has seen an exponential increase in data acquisition, and tools capable of effectively analyzing and interpreting these data have become a necessity. Whilst machine learning and artificial intelligence (AI) are widely used words, few people truly understand their implications or capabilities.

Neural networks (Figure 6) are one form of AI that is modelled based on the brain structure, being made up of a series of nodes (neurons) and pathways that connect them to one another. These nodes are organized into a number of layers, with the precise number of layers and neurons per layer being selected purely as a matter of choice of its creator or the complexity of the task it is intended to tackle. At its core, a neural network takes a set of numerical data into its input layer of nodes and produces a new set of numerical data via its output layer. Various internal algorithms are used to determine the output set based on the input set, but the precise inner workings of these networks are ultimately determined by how they are trained.

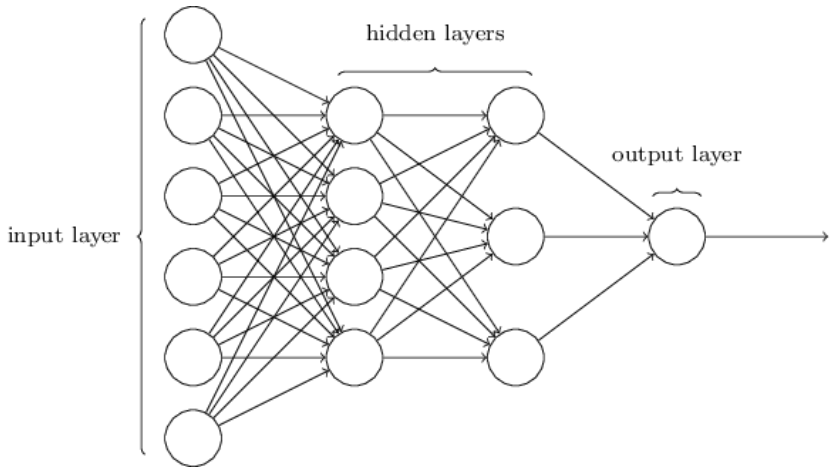


Figure 6 - Architecture of a neural network from Neural Networks and Deep Learning. Nielson, Michael A. Determination Press. 2015.

Before a neural network can be used effectively, it must be fed a large amount of training data. These data are used to alter various weights in the network's algorithms by telling it what output values should be achieved based on certain input values. Consider a neural network that functions as an 'AND' operator. This operator looks at two Boolean (true or false) values, and supplies a Boolean result based on those values. The AND operator works as follows:

- True AND True = True
- True AND False = False
- False AND True = False
- False AND False = False

A network based on these rules would have two input nodes and one output node, and can be trained using data in the format: (input, input : output). If true and false are substituted for 1 and 0 respectively, the network can use (1, 1 : 1), (1, 0 : 0), (0, 1 : 0), and (0, 0 : 0) as training data. Once trained this way, the network will function as an AND operator, supplying the desired output based on the inputs supplied to it.

The AND operator example is simplistic as it deals in Boolean or binary values, but it provides an idea of how a basic predictive neural network functions. A better example could be to feed a neural network training data in the form of images of cats and dogs, being sure to include an indicator of which animal is depicted. In this case, the network will have two output nodes; one for cat and one for dog. An output with values 1-0 would represent a dog, while 0-1 represents a cat. The input image data would have to be broken down into the red, green and blue numerical values for each pixel. After training the network on a large set of cat and dog images, any input image thereafter would be classified as either a cat or a dog.

The output values for neural networks are also generally a lot less binary. For the cat and dog example, the output data would often end up looking more like 0.913-0.087 in a case where the network is 91.3% sure that a dog is being shown in the image. These kinds of output values are known as 'confidences' and will always add up to a total of 1, irrespective of the number of output nodes (assuming the ReLU activation function is being used in the network, but this will be discussed later).

In order to further explain how a neural network is trained, the means by which the neural network evaluates itself during training must first be examined. A loss function is a mathematical equation used to determine the inconsistency between predicted values and training label values (Changhau, 2017), and this function is used throughout training to calculate how far off the mark the network's predictions are. The output of a loss function is always positive, with higher values indicating that the network's predictions are incorrect or inconsistent.

The loss functions seen in Table 1 are by no means the only ones that can be used, but are some of the more popular variations. Each has its own advantages and disadvantages that tailor its use to a particular kind of problem, making choosing the correct function extremely important in

order to achieve a trained network of optimal accuracy.

The loss function that will be the primary focus during the course of this project is the Cross Entropy. This function is most commonly used for binary classification (Changhau, 2017), which is precisely how the neural network in this project should function. Out of all of the physical stances of subjects that the network will be trained to recognize, it should be able to clearly single out one as the most likely choice at any point in time (output value of 1 for that node and 0 for the rest). While achieving a true binary output is almost impossible, this loss function should provide the best means of training the network to reach that goal.

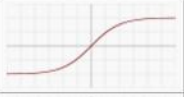
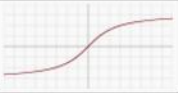
Table 1 - Loss functions table from the Theoretical Foundations of Machine Learning conference paper, On Loss Functions for Deep neural Networks. Janocha, Katarzyna and Czarnecki, Wojciech M. 2017.

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y}-\mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y}-\mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y}-\sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y}-\sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \lvert\sigma(\mathbf{o})^{(j)}-\mathbf{y}^{(j)}\rvert$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
$\log^2$	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2^2+\|\mathbf{y}\|_2^2-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log \frac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2\|\mathbf{y}\|_2}$

The activation function is another equation that handles the process of calculating the final output that is evaluated by the loss function in training, or used by whatever application the trained network is linked to (Sharma, 2017a). As is the case with loss functions, each activation function has its own set of advantages and disadvantages that tailor it for use in particular situations.

The two main types of activation functions are known as sigmoid and ReLU (Rectified Linear Unit), and can be seen in Table 2 as the S-shaped and more linear graphs, respectively. In all cases, x is the value being passed into the output node from deeper in the network. The core use of an activation function is to tailor the network's output to a particular format (Sharma, 2017a), such as binary or as a range between -1 and 1.

*Table 2 - Activation functions from a Towards Data Science article titled Activation Functions in Neural Networks. Sharma, Sagar. 6 September 2017. https://miro.medium.com/max/700/1*p_hyqAtyI8pbt2kEl6siOQ.png. Accessed 5 July 2018.*

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

When working with probabilities in a network's output layer, the activation function that is chosen must restrict the output range to values from 0 to 1, as 1 is the equivalent of 100% probability and values cannot be negative. The activation function that best achieves this output

format is ReLU as it scales linearly from 0 to 1 and has an output of 0 for all negative input values; consequently, this function will be the focus for this project's neural network.

The process of inputting data to a neural network and having it run through the various node-based calculations to achieve a final output is known as forward propagation. When training a neural network, a somewhat reversed process is also necessary. This process is called back propagation and relies heavily on the output of the loss function after each occurrence of forward propagation.

Once loss has been calculated, the network essentially reverses direction and begins adjusting variables known as weights. These weights are the defining factors that alter the calculations performed at each node in the network's structure, working in unison to reshape the input data to a prediction. As an increasing number of forward and back propagation iterations are performed, the weights for each node are adjusted in such a way that the network's predictions become more and more accurate, decreasing the loss value (Moawad, 2018).

The process of passing an entire set of training data through the network in this manner is known as an epoch. Training over multiple epochs steadily alters the network's weights to better suit the training data, and as such, a higher number of epochs increases classification accuracy further, up to a point. Training over to many epochs can cause 'overfitting', where the network's weights become so tailored to the training data that it can no longer accurately classify anything that falls outside of the range of that data, even by a small degree (Sharma, 2017b). This causes the network to act like a lookup table rather than a predictive tool.

These predictive neural networks are exceptionally useful as they are capable of accurately classifying things that they have not necessarily seen before, so long as they can recognize an overarching pattern. This makes them ideal for gesture recognition as they focus on the pattern that defines the gesture, rather than the size or shape of the person performing the gesture. Leveraging this ability should provide accurate gesture recognition for each subject, irrespective of whether the network was trained using a body type similar to that individual.

3. Methodology of Practical Work

In order to test various motion capture and neural network combinations, a modular project was developed in the form of a game. This game is based on the universe of Nickelodeon's (2005-2008) *Avatar: The Last Airbender*, and pits players against each other in a one-on-one elemental 'bending' fight. These players perform gestures based on one or more form of martial arts in order to create offensive and defensive effects, such as hurling rocks or fireballs or blocking projectiles with shields made of air or water. The physical techniques and abilities shown in *Avatar: The Last Airbender* were selected due to how well those physical movements match the effects they create, as well as how visually satisfying and physical the effects themselves look and feel.



Figure 7 - Firebending examples from an Avatar, the Last Airbender poster. Daljo. https://i.pinimg.com/564x/c4/b2/88/c4b288bdda7e24d8bcb73ee2fbcf0cca.jpg. Accessed 12 October 2019.

The different elements that can be controlled are associated with different forms of martial arts, with each form mimicking the characteristics of its element. The project begins with the slowest, most rigid form - earthbending. The gesture set that controls the earth element is modelled on Hung Gar Kuen, a southern Chinese martial art that places particular focus on strong, low stances and powerful punches (Roussea, 2017, Wing Lam Kung F, 2011). This should provide the best baseline from which to gather data before moving on to gestures that are faster and more fluid. Time permitting, the intention was to then add each of the other 3 elements and their respective martial arts. Fire, water and airbending are based on more fluid martial arts, making them more accurate representations of natural movement. This fluidity also makes them more difficult to gather data from as individual stances are held for much shorter periods of time. The project was originally designed to be considered complete once the system was functionally simulating earthbending, but the addition of the remaining three elements was considered to be attainable provided there were no major unforeseen issues.



Figure 8 - Earthbending examples from an article on martial arts influences in Avatar, the Last Airbender. JaketheSnake486. 5 October 2014. https://i.imgur.com/tgpfbnD.jpg. Accessed 15 July 2018.

While presence is very subjective and thus cannot be quantified, the purpose of this project was to explore tools that enable presence while gathering immersion metrics, namely the accuracy of movement capture and the frame rate at which the system is able to operate. The decision for the game to take a multiplayer format rather than pitting a single player *vs* AI was made in order to further enable presence through a sense of copresence, creating an environment where every aspect of the game is controlled by a human player, especially the movement of the character meshes. The mapping of the motion capture data to these meshes is of the utmost importance, as elements such as jitter and occlusion easily break the player's feeling of presence.

The motion capture element of the project was achieved using one or more Kinect sensors, which are primarily built to track the human body and extract basic skeletal structure data from the images captured using infrared and colour cameras. These sensors were tested in multiple setups with the goal of determining which setup provided the best captures. The data provided by the sensors comes in the form of vector coordinates for the various bones on the Kinect skeleton, as well as their rotations and relations to one another. Using multiple sensors to collect these data allowed for the generated skeletons to be overlaid on one another in the hopes of creating a mean version that is more accurate than each individual skeleton.

The data points provided by the skeleton are as follows:

- Leg
    - Hip
    - Knee
    - Ankle
    - Tip of foot
- Arm
    - Shoulder
    - Elbow
    - Wrist
    - Tip of index finger
- Spine
    - Pelvis
    - Lower back
    - Base of neck
    - Head

Features were then extracted from these data points in the form of distances calculated between each point. For example, if a frame of capture provides positional data for 4x2 points for legs, 4x2 for arms and 4 for the spine, there are a total of 20 points throughout the body. Using the formula n(n+1)/2, where n is the number of points, 190 distances are calculated per frame. The reason distances are used as features rather than raw positional data is to allow for gestures to be performed at any location within the field of capture and at any angle without skewing results. Distances are also much easier to normalize when accounting for players of varying heights, rather than having to adjust positional data to fit a mean.

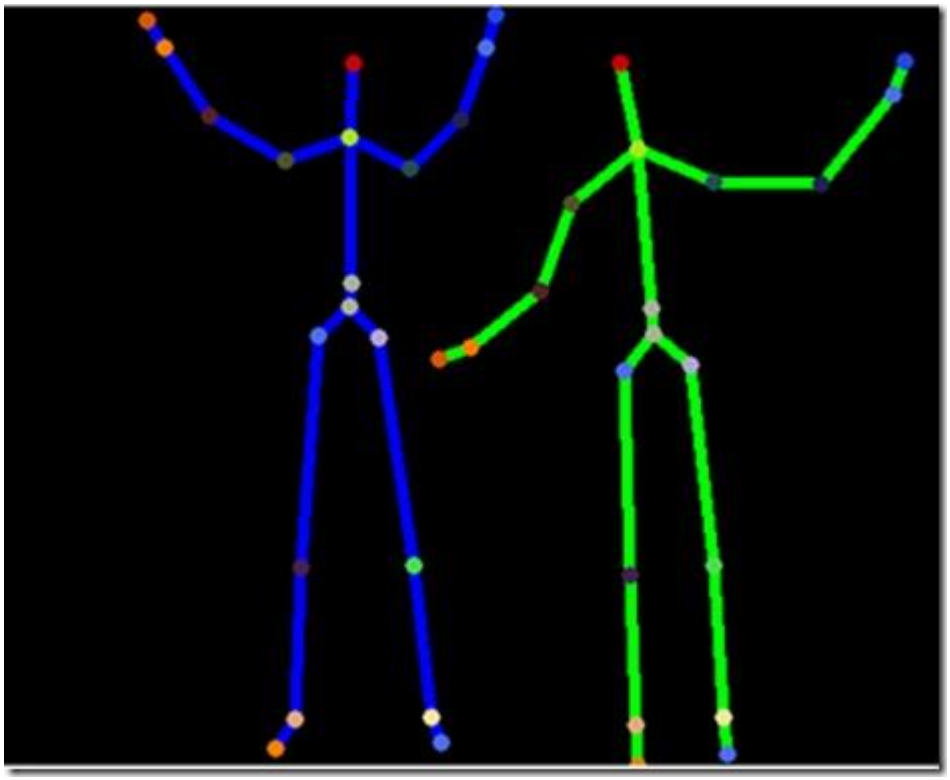


Figure 9 - Kinect skeletons from a Code Project article, Kinect – Getting Started – Become The Incredible Hulk. Raiten, Shai. 18 June 2011. https://www.codeproject.com/KB/dotnet/KinectGettingStarted/8.png. Accessed 15 july 2018

Each complete gesture or movement that triggers an action in the game was broken down into multiple stances from start to finish. Each of these stances needed to be moved through, in the correct order, to register a completed gesture. In order to train a neural network to recognize the stances that make up each gesture, many frames needed to be gathered from a group of people all holding the required stances. Hundreds, if not thousands of captures were needed for each stance performed by each person for each gesture, and each of these captures then had to

have distance data extracted from them. As confirmed by testing using a prototype of the system, this process required no more than ten seconds of capture per stance, per person.

The distance sets for each stance were fed into the input layer of a neural network for training, along with an identifier designating the node in the output layer that should be triggered. After enough training, the neural network should be able to classify distance data fed to it, determining which of the stances it was trained on is the most likely candidate. The game would then be designed to constantly stream distance data to the neural network, classifying the best-fitting stance for each new movement made by the player in real-time.

In order for the system to recognize a full gesture, the player has to move through that gesture while the neural network attempts to match each captured frame to a stance that it was trained on. If all of the stances that make up the gesture are correctly classified with a reasonable degree of certainty and in the correct order, the gesture will register as completed and the appropriate effect will occur in the game.

To further ensure that gestures are not incorrectly registered, time windows between stances were implemented. This ensured that players were not able to perform the first half of the stances for one gesture, a full set of stances for a second gesture, and only then return to complete the last few stances for the first gesture. A threshold value was implemented to minimize false positives occurring while a player was merely dodging attacks or idling. Varying degrees of thresholding were also used to track how accurately stances were performed, using this accuracy to affect the power level of the abilities triggered by the gestures that these stances make up.

As stated previously, the project adopted a modular design. This means that while the core game remained the same, the methods of input and classification could be easily swapped in and out. A single Kinect sensor was used for baseline testing. It was set up in front of the player at a height of one metre off the ground and two metres from the player. This was followed by two sensors in front of the player, one shifted slightly to the left and the other slightly right, both rotated to focus on the capture location two metres away.

The second modular element in this project is the neural network, where the baseline tests were

initially performed using only 2 layers - input and output. As mentioned previously, the network used the ReLU (Rectified Linear Unit) activation function and the Cross Entropy loss function throughout its five epochs of training. From there the network was trained on datasets captured from individuals, multiple datasets based on small pools of people grouped by height, and broad datasets made up of all available training data. These tests were intertwined with the Kinect setup tests and each combination had its responsiveness, classification accuracy, and stance replication examined as metrics for comparison.

Two main options exist for handling the display element of the system, namely free camera movement based on VR headset sensors, and a locked or semi-locked camera that always keeps the opposing player in view. While free movement is sure to be the better enabler for a presence to be experienced, excessive turning causes issues with cabling. Completely locking the camera to always be centered on the opposition greatly limits feelings of presence, but does heavily disincentivize turning. The best option was to limit the camera to face forward with up to 45 degrees of turning allowed in any direction. This created a cone of possible view, allowing the player to examine the environment in front of them while still disincentivizing turning and negating cable issues.

After all tests were completed, a single combination of Kinect sensor setup and neural network training should emerge as the most viable option for controlling games and other such media, providing the best balance between gesture recognition accuracy and processing efficiency, measured as frames per second. Based on previous experience, it was anticipated that a combination of more individually-focused neural networks and a larger number of sensors would provide the best results.

4. System Implementation

The first step when building this system was to determine all of the elements that would need to function in unison to achieve a baseline for simulating elemental bending in a believable and practical manner. At its core, the system needs to capture skeletal data via at least one Kinect in order to determine whether the skeleton depicts a particular stance. Adding VR functionality is

the next step, followed by networking elements to allow for multiplayer involvement.

Owing to how heavily the project relies on accurate motion capture to achieve body-matching and gesture recognition, the logical starting point was to explore existing Kinect-based projects within the Unity engine. While a few were found, it was interesting to note that not a single one of them made use of Microsoft's Kinect SDK (Software Development Kit) to handle skeleton extraction. Each of these projects instead accessed Kinect data by viewing the sensor as a generic peripheral device and using its video feed to build their own skeletal representations of the capture subject.

After weeks of testing various versions of Unity, the Kinect SDK and the .Net framework, it became apparent that the reason the existing Kinect projects did not make use of the supplied SDK is that it is not compatible with Unity's build system. Whenever a build of the project's code was being compiled, the Unity engine would remove the reference to the SDK and then be unable to access the functionality it was intended to provide. Another method of leveraging the SDK was attempted, this time using the NuGet plugin for Unity to access the Kinect library as a NuGet package. The issue persisted as Unity continued to automatically remove the package whenever code was compiled. This presented a large issue that would need to be handled at some point, but at this early stage it was more important to get the base system built as soon as possible. Any research done into why the SDK would not be accepted by Unity was met with many others asking the same question and no definite answer. Fortunately, the code from these existing projects could be adapted to work with this system provided that only one Kinect sensor was being used.

4.1. Threshold Stances

With skeleton data now easily available, the next task was to use that data to create 'earthbending' effects, such as pulling rocks out of the ground and sending them flying through the air. As mentioned previously, the premise of earthbending is based on the Hung Gar Kuen martial art form, which is defined by its long, powerful stances. As such, basing the system on earthbending was an ideal choice for multiple reasons: it utilized an array of steady stances that were both easy to learn and capture, and provided the user with a very satisfying game feel when they mastered

the art of throwing huge rocks around the environment.

Making use of every stance in the Hung Gar Kuen arsenal would have made capture and recognition very difficult, thus a small pool of fixed stances were selected that felt appropriate for the chosen earthbending actions, namely stomping the ground to lift rocks and then either punching or kicking to propel the rocks away from the player. The skeleton data supplied by each frame was used to calculate distances between key joints, which were then displayed. These distances were then used to map out the chosen stances using threshold.

As an example, the player would be considered to be punching with their right hand if the distance between the right wrist and shoulder joints exceeded a certain value, and if the difference in height between them did not exceed a certain value. This ensured that the player's arm was extended and roughly angled along the horizontal plane. This technique explained in this height and extension example was also applied to the left arm and legs so that punching and kicking could be recognized on either side of the body.

Unfortunately, triggering bending actions based on single stances like this failed to force the player to actually punch or kick to perform these actions. Following the right punch example, if the player had their right arm fully extended at their side and then proceeded to rotate their shoulder until they reached the threshold stance, they would trigger the punch effect by swinging their arm rather than actually punching as intended. To combat this, more stances were added to act as windups before performing the trigger stances.

In order to perform an action such as the right punch, a player would now have to first reach the windup stance of having their right wrist and shoulder joints be within a very small distance of each other. Once this stance was reached, the system would then begin looking for the previously mentioned threshold values that represented the punch. This dual stance setup proved to be very effective at forcing players to perform the physical movements for an action as intended, and it was used to achieve fluid punching, kicking and stomping actions.

The major downside to using threshold values to determine what stance a player is holding at any given time is that the player using the system may have very different proportions to the player whose body provided the initial threshold values. This became immediately apparent

when a new player could not extend their wrists past the threshold value needed to perform a punch stance as their arms were shorter than the baseline player's arms.

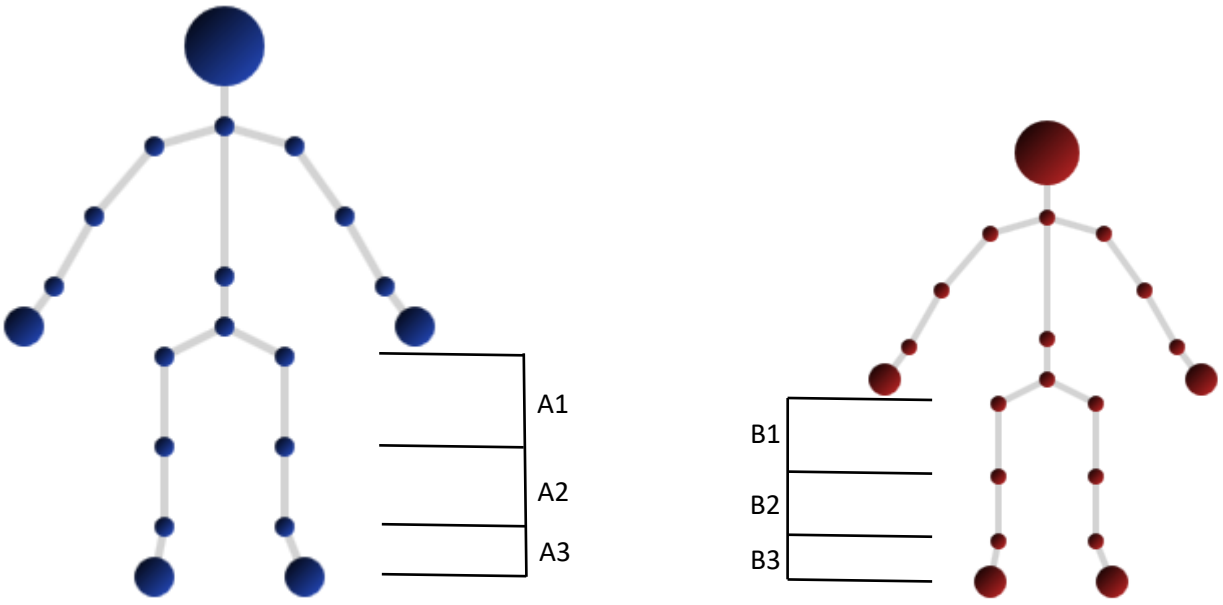


Figure 10 - Kinect skeleton height comparison

In Figure 10, a clear difference is shown in the distances between joints on the legs of different-sized skeletons. As an example, suppose a kick was defined as the knee joint being raised above a height of A2 + A3 + 0.5*A1, combined with the distance between the hip and ankle joints being 0.8*(A1 + A2) to show leg extension. The blue skeleton would be able to perform accurately classified kicks without issue, whereas the red skeleton would have to first raise its knee to hip height at minimum, and after that hope that the length of its leg was long enough to pass the 80% threshold applied to the blue skeleton's leg size.

Table 3 - Relative Body Measurements

Subject Height	**Leg Length**	**Arm Length**	**Shoulder Span**	**Head to Hips**
Tall	1. 022605	0. 6184507	0. 3857719	0. 8008837
Short	0. 7795032	0. 410091	0. 3070235	0. 3179092

Stances classified by thresholded distances, where those threshold and distance values are based on individuals of a certain size, are inherently biased. If the example had been swapped and the distances used were the B values instead of A, the problem would persist, though its manifestation would differ somewhat. The blue skeleton would barely be required to raise its knee at all, and its leg would only have to be extended slightly. This loosens the classifier to the extent that the action of simply standing, as shown in the figure, could trigger a positive kick classification.

At this point there were two major possible routes going forward: either adapt the threshold system to dynamically scale all baseline threshold values based on the proportions of the current player, or find a new way to classify stances that was more effective than basic thresholding. This is where neural networks become very useful, as they learn to recognize pattern during training rather than basing decisions on the raw values fed to them. It is important to note that thresholding was never the intended means of classification for this system, but its implementation needed to be used to show the contrast between the results it provided and the results provided by neural network-based equivalents.

4.2. Neural Network-Based Stance Classification

The first step in designing the structure of the neural network was deciding the features on which the stance classification should be based. The values for features would need to be consistent every time the player performed a given stance, irrespective of that player's position in relation to the sensor. This was necessary to ensure that players would be able to perform actions in any chosen direction, allowing them to aim the effects triggered by their stances. The only way to achieve such a feature set was to look at the positions of joints in the skeleton in relation to each other rather than the position of the sensor.

The distances between each possible pair of joints were calculated, yielding 190 floating point values to be used as values for the input nodes in the network. During tests from a number of angles and positions, these features maintained similar values each time a stance was repeated. The output layer of the network contained only 10 nodes, one for each chosen stance. A training

data capture system was developed to capture 1000 instances of skeleton data per stance from the player over a span of 10-15 seconds. 70% of these data was then fed into the network for training while the remaining 30% was used to evaluate the training process. This training was done using the previously examined ReLU activation and Cross Entropy loss functions. After five epochs of training, the network had achieved a state in which 99.41% of its stance predictions were correct.

When it came to testing this trained network as part of the main system, it was found that its classifications were actually very inaccurate, classifying most stances correctly only 50% of the time. Upon further examination, it was discovered that the player was no longer holding the same upper body stance when performing primarily lower body stances, and vice versa. For example, while a right punch was being performed, the left arm was in the same position as it had been during training, but the left and right legs were not.

This prompted another decision with regards to how stances are classified: should the system be trained more leniently to allow a large number of variations for any given stance, or should the stances remain as rigid as possible to ensure that they were being performed correctly? The first option would no doubt lead to accuracy issues down the line, but the second option was making accurate classification impossible in its current state.

A middle ground was created in the form of dual neural networks that could be run simultaneously, one to handle upper body classifications and the other to handle lower body. Owing to how the first network's input layer size was calculated, the first new neural network made use of only 36 input nodes as its focus was on only nine joints (pelvis and below). The kicking and stomping stances for each leg produced 4 output nodes for classification confidences, and an additional two were added to represent a standing stance and a squat stance. As for the upper half of the body, the eleven remaining joints produced an input layer size of 55 nodes, while the punch stances provided only two output nodes, supplemented with an arm tuck stance for punch windups and an arm raise stance, totaling just four stances overall.

It was apparent that the system was classifying stances both more accurately and faster than the single network setup. The change in classification speed was not surprising as the number of

calculations needed to perform a single classification on the single network was 1900 (190x10), whereas the dual network setup was only 436 (36x6 + 55x4). This setup yielded accurate classifications 98% of the time, making only two inaccurate classifications in a test where each stance was performed ten times.

4.3. Classification Implementation in Unity

The biggest issue when implementing neural network-based stance classifications came when the network had to be integrated into the Unity project built for the threshold stances. The first part of this problem was that the network code was written using Keras and TensorFlow in Python 3 whereas Unity projects are written in C#. This was solved by using the NNSharp library to store the trained weights and structure of the network as a JSON file which could then be used to run the network in C#.

The second issue became apparent shortly afterwards while trying to utilize the NNSharp NuGet package in the Unity project. Each time the project was built, the package would disappear, and all of the code associated with the functionality it provided became useless. The problem was almost identical to the one discovered when trying to utilize the Kinect SDK in Unity. This issue only occurred during Unity project builds, so a separate console application was created to test the package's functionality. While it worked perfectly in the new application, its functionality was critical for the Unity project for the system to be able to perform as intended.

One of the quickest means of transferring data between two applications is memory mapped files (MMFs). These act just like files stored on a computer's hard drive but are instead stored directly in memory. This makes reading and writing of data exceptionally fast, allowing for the possibility of multiple data transfers per second between applications. The process of streaming data between Unity and the console application required four MMFs, and worked as follows:

- Each new frame in the Unity loop extracts feature data from the Kinect skeleton and writes that data to the upper and lower body feature MMFs.
- Unity checks if the confidence data MMFs have been updated.

- If so, a visual effect is triggered to show the player that they their stance has been recognized.
- If not, no effects are triggered and the loop restarts.

- The console applications' respective loops check if their feature MMFs have been updated.
  - If so, they input those features into the neural networks and write the confidence outputs to MMFs. The loop is then restarted.
  - If not, the loop is simply restarted.

This idea did not work initially, however, owing to a signature lock built into the C# library that handles MMF functionality. When the Unity code created the feature MMFs, it marked them with a digital signature that prevented all other processes from accessing that file for reading or writing purposes. The exact same issue occurred with the confidence MMFs created by the console applications. The solution to work around this signature lock was simple; a new C# library was created to handle the reading and writing feature and confidence data to and from MMFs, and the functionality supplied by this library was then used in both the Unity and console applications to ensure that their MMFs all used the same signature.

While the constant reading and writing using MMFs may seem like a complex process that should slow down the system greatly, the addition of a system clock check to the Unity code showed that the confidence data MMFs were being populated with new data just over 100 times each second. Maintaining a high number of cycles is imperative to ensuring that the system can output a framerate suitable for virtual reality. If the framerate were to drop too low the player would begin to experience jitter, which both breaks the feeling of presence that this project was striving to achieve and places unnecessary strain on the eyes.

At this stage of implementation, questions were already being raised regarding how the transfer rate would be affected by the addition of more complexity to the capture process or classifications performed by the neural networks. It became immediately apparent that the addition of more stances or Kinects could slow the system down to a point where it was no longer

viable for VR.

4.4. Implementing Bending Effects

The easiest way to give a player feedback on whether or not they are performing the correct stances is to supply them with constant visual feedback. This initially took the form of raising rocks out of the ground after successful stomps and then pushing these floating rocks away from the player's avatar after successful punches and kicks. During initial tests, it was difficult to tell whether stance sequences were being properly captured and identified as many stances did not have their own indicators showing that they had been recognized. A punch stance causing a rock to fly away as if pushed was easy to see, but there was still no indication of whether the wind-up stance for a punch was recognized. This often brought about situations when intended effects were not achieved after performing the relevant two-stance sequence as the first stance in that sequence was not properly recognized. Reviewing the data revealed that this was either due to occlusions in the capture, as shown by the presence of default positional values, or the stance not being classified correctly by the neural network.

This issue showed the necessity for even more feedback. Consequently, many more effects were added to better let the player track the progression of their stance transitions. For instance, in order to make the height needed for leg-lifting wind-up that preceded the stomp stance more obvious, small rocks were scattered around the virtual environment. These rocks would rise higher in proportion to how much the player lifted their leg. Once the wind-up stance was recognized, the rocks would stop rising and start to rotate instead, providing the player with the feedback they needed to know that they could move on to the stomp stance. This method of feedback was added to the punches and kicks as well, causing the target rock to rotate once the relevant wind-up stance had been reached.

While implementing this feedback, a number of somewhat less generic effects were also developed for future stances. These included multiple walls of rock that the player could raise out of the ground in front of them to use as a shield, a shockwave that rippled along the ground from the player in a chosen direction, and a number of small rocks that floated in the air and then

combined to form a stone gauntlet that could be worn by the player's avatar and fired off, piece by piece.

These new effects brought about a new wave of stances needed to trigger them, increasing the total number from 10 to 25. Training data for these stances then had to be captured and used to update the neural networks. This addition caused the networks' classification accuracy to drop to 81%, which was not surprising considering the similarity of some of the new stances. As an example, consider the two simple neural networks shown in Figure 11. The structure of neural networks is one in which each node in one layer is connected to each node in the subsequent layer, making the total number of connections between each layer pair equal to the product of the number of nodes in each of those layers. Each of these connections holds a weighted value used in the network's classification algorithms, and each of these values is created using back propagation during training. When adding an extra output node or, in the case of this system, an additional stance to be classified, the number of weights that need to be trained increases by the number of input nodes.

Each time an output node is added to include a new stance, it becomes increasingly difficult to adjust all weights during training to a point where the network can utilize them to perform accurate classifications. This issue is made even more prevalent when the new stance is similar to an existing one, causing further confusion when the network attempts to adjust its weights in such a way that will aid in differentiating between the two. With this in mind, a decision was made to retrain the networks without the shockwave or gauntlet stances, which were too similar to many of the original stances. This change brought the total number of stances down to just thirteen and the classification accuracy back up to 98%.

After further testing, it was found that three of the stances were being incorrectly classified as each other owing to their similarity to one another. They were thus removed from the stance set, returning to the original lower body set of leg raises, kicks, stand and squat stances, and the upper body set of punches, tuck and arm raise stances. The tests performed during the training of these networks yielded an accuracy reading of 100% from as early as the third epoch.

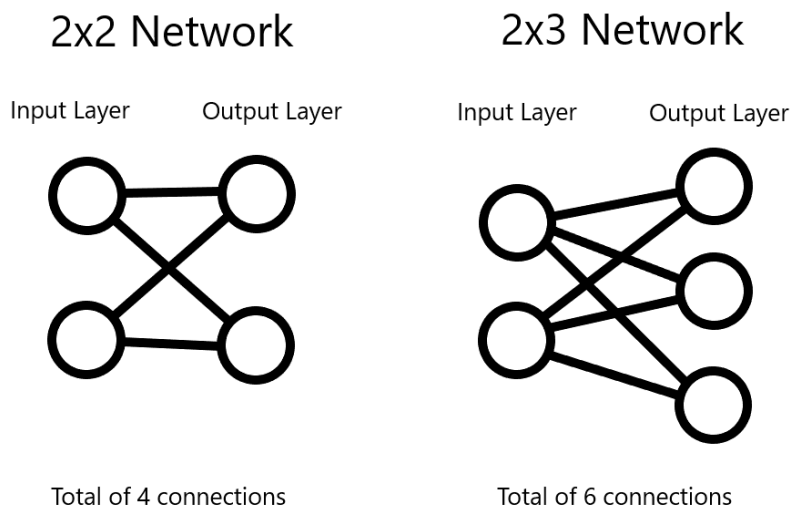


Figure 11 - Node addition in neural networks

4.5. Utilizing Multiple Kinect Sensors

While the 100% accuracy during training seemed very promising, the single sensor setup had one major issue - occlusions. Occasionally the player would get a delayed effect trigger on a punch or kick stance towards the sensor as the joints in the player's active limb would line up almost perfectly with the camera. This caused issues where the hip, knee and ankle joints could not be seen through the foot joint during a kick, or the shoulder, elbow and wrist joints could not be seen through the hand joint during punches. With these joints missing, features could not be accurately extracted, and the neural networks would not be able to classify the stance until the angle of the punch or kick changed enough for all joints to be seen.

This problem prompted the need to revisit the implementation of multiple Kinect sensors, using their combined skeletal data to create a more accurate 'mean' skeleton that would negate any occlusions on one of the sensors. As shown in figure 12, the red circles represent joints that are occluded by the white joint in front of them. When multiple Kinects are used to capture a stance, the offset of the second Kinect causes it to avoid any occlusion issues and capture all joints.

The joint position data acquired from each Kinect were examined to determine whether any occlusions were occurring in any given capture. If joints were missing owing to occlusions on

Kinect A, the mean skeleton updated its joint positions using the data for those missing joints acquired from Kinect B. If both Kinects captured any given joint, the average position for that joint was calculated from the two captures and the mean skeleton was updated accordingly.

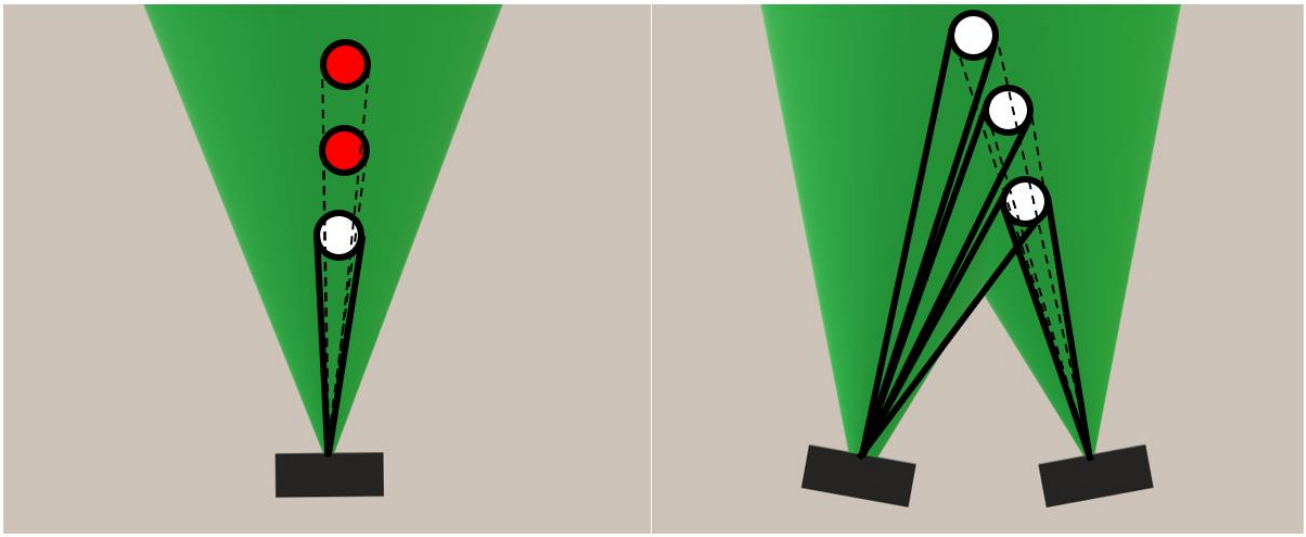


Figure 12 - Single vs dual Kinect capture to address occlusion issues

As stated previously, the Kinect code used up to this point was taken from an asset store project, *Kinect with MS-SDK*, and adapted. Unfortunately, its code was built to utilize capture from a single Kinect and could not be adapted to handle multiple sensors. Owing to this, along with the fact that the Kinect SDK could not be used in Unity, the need for yet another external application arose in order to facilitate communication between Unity and the sensors.

The current project setup was already making use of console applications to run the neural network and stream data to and from Unity, so the initial idea was to utilize the Kinect SDK in the existing console applications. Unfortunately, this did not work as each sensor could only be accessed by one application at a time, so either one application would have access to both sensors while the other had access to none, or they each had access to only one. Neither of these situations provided a solution to the problem, so a third console application was created to handle inputs from both sensors.

This application bypassed Unity entirely, calculating a mean skeleton, extracting upper and lower body features and writing them to MMFs just as Unity had done previously. The initial assumption

was that this change would negatively affect the system by reducing the number of loops each second, but the effect was the opposite. This was because Unity no longer had to handle any Kinect code, so its only task was to update the visual effects based on triggering stances. As for the console applications, the number of classifications they performed each second also increased as the new Kinect data application's loops were faster than Unity's owing to them not being tied down by the hidden processes contained within each Unity loop. This meant that Unity was now able to perform all of its usual frame-by-frame calculations involving elements such as object physics and scene lighting without also having to extract positional and rotational data from the images that were previously being streamed in by the Kinect code.

The only downside to removing the Kinect feed from Unity was that the character avatar was no longer being fed the data it needed to represent that player. Yet again, the custom MMF library was updated to allow the Kinect application to send raw joint position data to Unity, providing the avatar script with the data it needed. This change had the possibility of slowing down the loops in both Unity and the Kinect console application as they were now required to essentially pass a communication baton between the two of them, but the effect of this addition was found to be negligible in both cases.

It was further discovered that the Kinects did not require to be positioned very far from one another in order to overcome the occlusion issue. As occlusions only occurred when a limb was pointing almost directly towards the sensor, the second sensor needed to be placed only a metre to the left or right in order to cover the missing joints. This meant that both Kinects could still face the player from the front as opposed to a more side-on approach, providing the best capture view while maintaining optimal accuracy.

4.6. Player Avatar and Body-Matching

The meshed avatar used to represent the player in this project was particularly important as failure to match its movements to the player could result in an immediate break in presence. The Unity project included scripts that could be used to animate humanoid avatars using Kinect data, but the body-matching it achieved was very inconsistent. For instance, the avatar's arm

movements would closely match those of the player until a punch stance was performed, at which point the extended arm would be angled further outwards, away from the player's chest. This problem persisted when the player touched hands in front of their chest or face, with the avatar still showing the hands a fair distance apart.

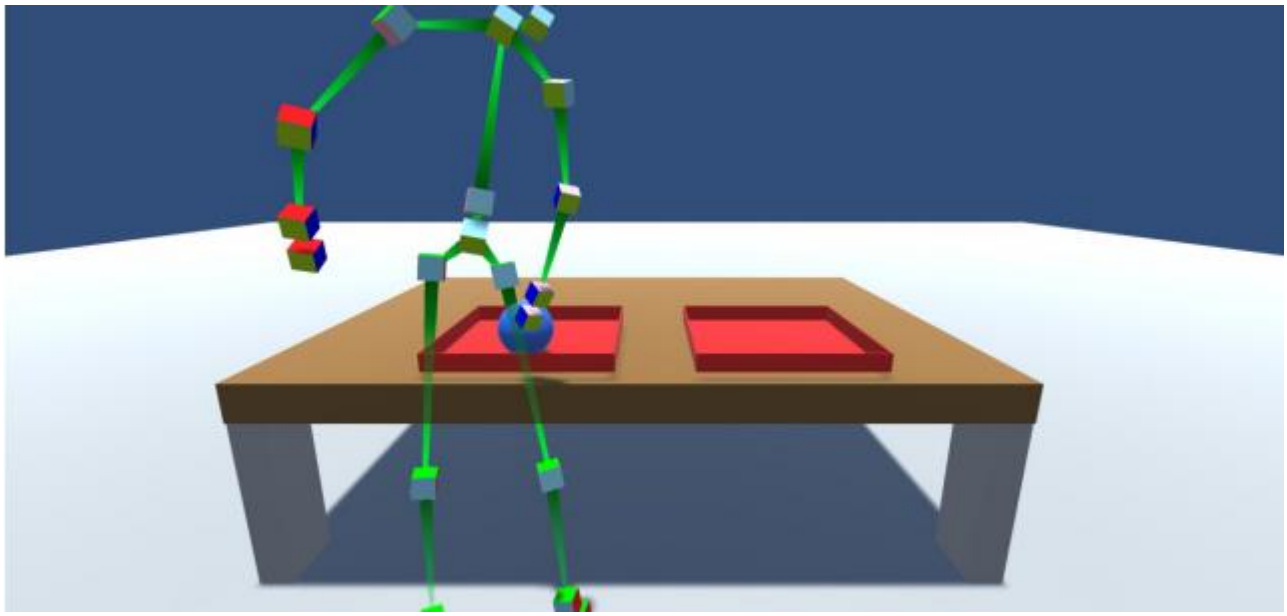


Figure 13 - Kinect cubeman skeleton

This behavior was strange as the placeholder avatar did not act in the same way as any of the meshed ones, maintaining accurate body matching throughout tests. This placeholder avatar was made up of floating cubes positioned at each joint location and lines connecting these cubes to form the bones in its skeletal representation of the player, as shown in Figure 13. After placing the 'cubeman' avatar and the various meshed avatars in the same space, it became apparent that all of them had different proportions. An examination of the project's code showed that the meshed avatar would need to be tailored to fit the proportions of the player in order to achieve an accurate representation.

Considering that the use of well-trained neural networks for classification allows the system to be used by people of a wide variety of body shapes, this body-matching system would need to be adapted to scale the avatar's proportions dynamically, based on the player's proportions. A second option would be to create a number of avatars with slightly different proportions and determine which one would be best suited to any given player.

While the second option would have been easiest to achieve, players and their best-matching avatar would almost never completely match, leading to the same problem. While the mismatch would be much less obvious, the goal of this project is to create as accurate a match as possible, and thus the dynamic scaling option needed to be implemented. In fact, Unity made this much easier than anticipated, using the already calculated distances between joints to scale the relevant transforms within the avatar. This process also added no overhead to the standard loop as it only needed to be run once per player.

A second issue with body matching was the lack of rotations. The player would rotate the wrist to move the hand from a palm-down to palm-up position, but the avatar would retain its hand's default orientation throughout. This presented a serious problem, as without the nuanced rotations that occur naturally in everyday movement, the avatar's movement felt awkward and forced. Delving deeper into the Kinect SDK yielded a possible solution to this issue in the form of a list of estimated quaternion rotations for each bone in the Kinect skeleton. The MMF library was updated to allow rotation data to be added to the positional data already being sent to Unity.

This rotational data proved to be far less useful than originally anticipated, as it was found to be calculated based on relative joint positions rather than the capture of any of the user's actual rotations. For instance, the rotational data of an elbow bend would be represented almost perfectly because the change in angle was being calculated using the change in position of the wrist joint relative to the elbow and shoulder joints on that arm. A palm up to palm down rotation, on the other hand, showed no change in relative joint positions, and thus showed no rotational change.

This problem proved to be more of a minor inconvenience than a true presence-break owing to the way transforms function in Unity's GameObject hierarchy. When one object is assigned to be a child of another, it maintains its position and orientation relative to that parent. As such, if a child object is located one unit of distance below its parent and that parent is rotated by 180 degrees, that child will be shifted to one unit above the parent. Organizing the Kinect skeleton joint objects in a hierarchical fashion proved to shore up most of the previous rotational issue, though the system was still far from true rotational accuracy.



Figure 14 – Rock golem used for body matching

A simple, somewhat abstracted character representation was decided on, inspired by the idea of the player's avatar being a 'golem' made up of the same kind of rocks that it was manipulating, shown in Figure 14. This made excellent use of the previously measured distance between joints to situate elongated rock objects at the centre points between joint pairs, forming rock bones that were added to the existing skeletal hierarchy so that they would shift and rotate accordingly.

4.7. Moving to Virtual Reality

As the system does not require handheld VR controllers, there was surprisingly little work needed to implement VR into the existing project. As the HTC Vive was being utilized, which runs using Steam VR, it was a simple matter of importing the Steam VR assets from the Unity Store, updating project settings and adding the 'Tracked Object' script to the main camera attached to the head of the cubeman avatar.

This caused an issue where the player's head movement was essentially doubled as the HMD camera was being moved to match the cubeman's head position and then moved again to account

for the position update observed by the Vive sensors. This posed a problem to which there was no immediate answer, as removing the Vive tracking would yield the best positional relationship between the avatar's head and the rest of its body, but would make head rotation tracking much less accurate without. To overcome this, a script was created to reset the camera's position each frame. This allowed the Steam VR script to provide the much-needed rotation accuracy without the Kinect rotations causing any extra head movement. However, using the Vive's tracking rather than that of the Kinects created another issue - The HMD movements captured by the Vive's lighthouses did not correspond exactly with the full-body movement captured by the Kinects.

This positional mismatch between the sensors caused the avatar's head to separate more and more from its body the further the player moved away from the centre of the capture area. This created a somewhat out-of-body experience that was a definite break in presence. An attempt was made to use the skeletal hierarchy yet again to solve this problem, but making the HMD the topmost parent object caused the entire skeleton to rotate with it on all three axes when the player looked around.

As a result of these problems, a different approach was needed that did not rely on the use of Unity's object hierarchy. The new fix involved calculating the positional offsets between the head joint and each other joint in the skeleton, and then applying those offsets to the HMD position in each frame to calculate the new positions for each joint. This essentially replaced the head joint with the HMD and ignored the pure positional data provided by the Kinects in favour of a more relational position model. The fix achieved the desired effect, moving the avatar around the capture space as dictated by the HMD's more accurate movement.

4.8. Revisiting Thresholding

While neural networks are extremely efficient at performing classifications, the process by which they make those classifications is a computationally intense one. Considering that a VR system needs to run at a minimum of 90 frames per second in order to avoid flashing in the user's peripheral vision, the use of neural networks had a high chance of reducing the system's feasibility. This prompted a return to the idea of thresholding stances.

The biggest issue with the previous attempt at using thresholding to classify stances was that it did not scale well with a variety of body sizes. Overcoming this problem became the core focus of a new set of calculations that would be able to provide accurate stance classification regardless of the subject's size. This new set of algorithms focused more on ratios than on predefined values, using measurements taken from the subject's body every frame in order to tailor classifications to their own unique shape.

For example, when classifying a stance where the subject raises their knee to roughly hip height, bent at 90 degrees so that their lower leg is perpendicular to the floor, the system starts by checking the difference in elevation between their hip and knee. This difference is then compared to the distance between that hip and knee, the length of which is the same as the difference in elevation between the two while the subject is standing with that leg's foot on the ground. With those two values acquired, a knee raise can be classified as the subject having reduced the elevation between their hip and knee to below a certain percentage of the distance between the two.

This percentage value is what is known as the threshold. It determines how strict or lenient classification is for each stance. Certain stances may be very similar to one another, in which case stricter thresholding values need to be used in order to ensure that they are adequately defined in their own rights. The other way to further separate the definitions of similar stances from each other is through the use of more thresholding comparisons.

Continuing with the knee raise example, if the subject was to extend their leg parallel to the ground it should be classified as a kick rather than a raise. The issue in this case is that the knee raise and kick cannot both be defined purely by the hip-to-knee elevation-to-distance threshold, so an additional threshold is needed to differentiate between the two. Considering that the major difference between the two stances is the position and orientation of the lower leg, the change in distance between the hip and foot provides all the information needed to separate the two.

Instead of using elevation in this case, a sum of distances is used. When standing or kicking, the distance between the subject's hip and ankle is the same as the sum of the distances between their hip and knee, and their knee and ankle. When performing the knee raise stance, however,

the distance between the hip and ankle is drastically reduced while the sum of the other two distances remains the same. Knowing this, differentiating between the two stances is a simple matter of defining the threshold percentage that marks the difference between the knee being bent and extended.

Defining multiple thresholding checks for various limbs has the added benefit of inadvertently creating definitions for more stances. With a knee raise stance defined by the knee being elevated past a certain point while the hip-to-ankle distance is below a certain ratio, and the kick stance defined by the hip-to-ankle distance being above that ratio, the standing stance can be defined by the kick's hip-to-ankle side of the ratio and the knee being dropped below a certain point.

Applying these calculations across select joints on the body creates significantly less overhead than the calculations needed for classification in the neural networks. This method of stance definition has the highest chance of maintaining the 90 frames per second needed for a viable VR system, although the tradeoff between speed and accuracy has the potential to adversely affect that viability.

4.9. Networking and Multiplayer

As mentioned in the previous section, many concerns already existed regarding the computational intensity of the system, and thus the idea of a multiplayer system had to be discarded. With Unity already interacting with multiple external applications in order to manage Kinect capture and neural network-based classification, adding in an additional interaction with another device experiencing the same issues would have resulted in VR becoming completely infeasible.

If the system was developed as initially intended, with the aim of two users fighting one another, not only would each system have to handle its own Kinect capture, body-matching and gesture classification, but it would also have to send its body-matching data to the other system and use the body-matching data received from that other system to create an accurate depiction of the

other user. With the minimum of 90 frames displayed per second required to prevent users from experiencing presence-breaking flashes in their peripheral vision, the system would simply be unable to handle the processing of all of this additional data while still maintaining such a high framerate.

This calls into question the topic of computer hardware specifications as devices with better hardware should be able to run the system at a higher framerate, and could thus accommodate the additional processing while still maintaining the required framerate. This project is a proof of concept that has, from the very beginning, been targeted towards the lower end of the hardware spectrum as much as possible in order to examine feasibility. There is more information to be gained by examining how well the system functions on lower-end devices, and thus more possibility for future improvements. The possibility of scaling-up the hardware used to run the system will be discussed further at a later point in this paper.

4.10. Stance Enrolment and Neural Network Training

For the neural networks to be able to perform accurate stance classifications, a large amount of labeled data was needed. These data were acquired by having 30 subjects perform each of the six lower, and four upper, body stances over a period of 1000 frames, captured first using a single Kinect and second with the dual Kinect setup. The subjects were told to hold each of the stances over the desired period, all while slightly varying the stance in order to provide a wider range of data points. The data from each stance captured was stored in a CSV (Comma Separated Value) file, with columns labelling the specific distance recorded between each joint pair and each row representing a new frame of capture. The final column in each file was a label of the gesture number that the row's data represented: 0-3 for the four upper body stances and 0-6 for the lower ones. Each of the 20 stance captures for each subject was stored in its own file, along with a 'size' file detailing the subject's leg, arm and torso lengths, as well as their shoulder span.

Once all 30 subjects had been captured, every size file was examined in order to find a minimum and maximum value for each of the four sizes recorded. The difference between the minimum and maximum sizes was divided into thirds; one bracket for shorter, one for average, and one for

taller individuals. A linear three-way divide was selected to create these brackets over the use of standard deviation, as it provided a uniform bracket size rather than a set heavily skewed in favour of the average bracket.

With the three size brackets in place, all subjects' size files were re-examined in order to place them in the bracket that best fit their sizes. With the subjects' data sorted, the data could then be compiled into six CSV files per stance; a short, average, and tall file for single Kinect and the same for dual Kinects. A master file was also created for each stance and Kinect setup, containing the data from all subjects for that gesture. These CSV files were then used as training data for 16 neural networks, yielding an upper and lower body master network, and the same for each size bracket, doubled for the two Kinect setups.

Upon reviewing internal test metrics acquired during the training of the networks, it was found that just under half of them had less than 60% accuracy when classifying stances, with many of them dropping as low as 50% as seen in Figure 15. Examining these networks further showed that each of them had been trained using data captured with a single Kinect and that, when a frame had been captured while one or more joints were occluded, the distance data relating to those joints showed a linear increase on each subsequent frame, often scaling up values that were usually under one unit to values of over 1000 units.

```
Epoch 1/5
17100/17100 [==============================] - 28s 2ms/step - loss: 1.2449 - acc: 0.4708
Epoch 2/5
17100/17100 [==============================] - 31s 2ms/step - loss: 1.1434 - acc: 0.4884
Epoch 3/5
17100/17100 [==============================] - 26s 2ms/step - loss: 1.1255 - acc: 0.4941
Epoch 4/5
17100/17100 [==============================] - 28s 2ms/step - loss: 1.1145 - acc: 0.5009
Epoch 5/5
17100/17100 [==============================] - 31s 2ms/step - loss: 1.1077 - acc: 0.5032
```

Figure 15 - Neural network training metrics before applying the zero fix

To correct this, all CSV files were examined once more, and all distances of over two units were uniformly replaced with zero. With the files cleaned of inaccuracies, the 16 neural networks were retrained, all showing classification accuracy of over 95% as shown in Figure 16. With confirmation that the zero-replacement solution worked as intended, the system's code was

adjusted accordingly to immediately replace values greater than two with a value of zero.

```
Epoch 1/5
17100/17100 [==============================] - 30s 2ms/step - loss: 0.5458 - acc: 0.8829
Epoch 2/5
17100/17100 [==============================] - 26s 2ms/step - loss: 0.2452 - acc: 0.9605
Epoch 3/5
17100/17100 [==============================] - 28s 2ms/step - loss: 0.1934 - acc: 0.9699
Epoch 4/5
17100/17100 [==============================] - 27s 2ms/step - loss: 0.1690 - acc: 0.9744
Epoch 5/5
17100/17100 [==============================] - 28s 2ms/step - loss: 0.1537 - acc: 0.9769
```

Figure 16 – Neural network training metrics after applying the zero fix

Following this, fifteen subjects were chosen to have personal neural networks trained on their individual data for later testing; seven from the average bracket and four from the tall and short brackets. These networks yielded the best training results owing to the reduced variance in their training data, with not one of the networks showing classification accuracy of under 99%.

5. Testing the System

In order to test the system's accuracy, a group of 30 subjects was needed. These subjects were divided into two equal groups: 15 pure testers and 15 of the original subjects who had already participated in capturing data during the enrolment phase. The reason for this split was to determine whether the networks were able to classify someone whose data it had never seen just as accurately as someone whose data was used in its training.

With the neural networks trained, six of them were used to first test each of the original subjects. This began with the single Kinect broad-spectrum master network, followed by the network applicable to the subject's bracket, then their own personalized network. The fourth and final test for these subjects was done using dynamic thresholding algorithms. As with the enrolment, this process was repeated with the dual Kinect equivalents of the networks. Each of these tests involved the subjects performing the ten stances for 250 frames each. Once all the original subjects' test data was recorded, the pure tester subjects underwent the same process with the notable exclusion of individual networks as none existed for them.

During each frame captured while testing, the system fed the subject's data to either the applicable neural network or the thresholding algorithms for classification. The value returned by either of these classifiers was the number of the gesture that was most confidently identified. For the neural networks specifically, if the confidence value for the identified gesture was above 90%, the system noted that frame as having a 'precise' classification.

Once all 1000 frames had been captured for a given stance, the number of normal and precise classifications for each gesture were each recorded separately, and then totaled. Additional information was also stored within each file pertaining to the number of frames that were successfully recorded over the capture period, as well as the amount of time taken to complete the capture. Once all 30 subjects' test data had been acquired, the datasets were averaged and tabulated for easy inspection.

6. Results

Tables 3-6 were created in order to best compare the results captured during the testing phase, one for upper body stances and one for lower body stances, both captured using a single Kinect, and the same repeated for the dual Kinect setup. Each of these tables contain values for the percentage of accurately classified captures over a period of 250 frames, per classifier, per stance.

Additional columns were added to the tables to give a single evaluation metric to each classifier in the form of an average accuracy across all of that classifier's gestures. The average time taken for each classifier to capture the 250 frames was also added, and used to calculate the average number of frames per second captured during that period. Finally, an additional classifier row, 'Bracket', was added to show the average values across the 'Short', 'Mid', and 'Tall' brackets.

6.1. Solo Networks

Theoretically, any classification system trained on a set of data with low variance should perform better than one trained on a group with more variance, when it comes to classifying subjects that fall within a low variance dataset. In the case of the neural networks trained solely on individual

subjects, it was assumed that this would be the case owing to the fact that the weighted values adjusted within the network's structure would narrow down to better recognize that subject's proportions and thus provide more accurate classifications for them.

This was not the case, however, with the 'solo' networks showing the worst performance across all of the tests. The main reason for this is the fact that the data used to train and test these networks was captured optically rather than physically. Optical capture devices are far more subjective than their physical counterparts such as accelerometers or devices that measure changes in rotation and extension. This subjectivity means that capturing a subject on one day, with a certain lighting, in a certain environment, dressed in certain clothing, could provide drastic differences if one or more of those variables are changed when capturing again on a different day.

The solo networks' inability to account for these variables took what was originally perceived to be their greatest strength and turned it into their greatest flaw, narrowing down their recognition spectrum too much and thus greatly reducing their classification accuracy. The networks with the next smallest training sets were the short bracket ones and, even with only five subjects populating the training data, these networks performed 11% better than the solo at bare minimum, and over 30% better on the upper end of the scale.

6.2. Bracket Networks

With enough variance in their training to account for more capture variables while maintaining a narrowed spectrum of subjects, the bracketed networks were on track to being the best solution to the classification problem. The average accuracy of these networks dropped below 80% in only two cases: the single and dual Kinect tests for lower body stances. This can yet again be attributed to the issue of variance, more specifically the fact that shorter subjects have less variance in the distance values captured from them as the maximum extension of their limbs is naturally shorter than that of the subjects in the other two brackets. This particular lack of variance ties in heavily with the degree of difference between the stances classified by the network, which will be discussed further at a later stage.

Table 4 - Bracket Thresholds

Bracket Height	Leg Length	Arm Length	Shoulder Span	Head to Hips
Tall	> 0.945382	> 0. 6090592	> 0. 3478557	> 0. 7803808
Short	< 0. 8068447	< 0. 5108783	< 0. 3028736	< 0. 3272791

On the opposite end of the scale, the tall networks were trained on seven subjects and performed noticeably better than their short counterparts in both lower body tests, reinforcing the theory that more variance in training yields greater accuracy in testing. This theory was further supported by the results shown by the mid bracket which, performed best on average, being surpassed by the other two brackets only in the single Kinect upper body tests and only by a margin of 2.5%.

The mid bracket networks were trained on the largest datasets of all the neural networks, aside from the master networks. The bracket contained 18 training subjects, providing it with ample variance in terms of both the physical sizes of the subjects and way that those subjects held their stances. With an average accuracy ranging between 86% and 93%, this network is certainly a viable classifier for the project's gesture set. The classifier scheme, however, needs to be able to accurately classify gestures performed by any individual to use the system, not just those whose size allows them to make use of the mid bracket network.

The bracket networks are a unified classification scheme, and as such they are only as accurate as the least accurate among them, namely the short batch networks. No matter how well the mid or top bracket classifiers perform, their accuracy means nothing to a user whose is placed in the short bracket based on their size. It is impossible to leverage the accuracy of the other two brackets to benefit them, as both of those brackets have been trained exclusively on datasets captured from larger individuals, so using either of them as a classifier would be even less accurate than using the short bracket networks.

6.3. Master Networks

The master neural networks are a catch-all solution, having been trained on the data of every subject regardless of size. These networks have by far the most variance of any of the networks tested. It was originally thought that this variance would cause the network to struggle to provide accurate classifications, having been spread too thinly across datasets that spanned too wide. The second concern was that, with so much data being used to train these networks, they would essentially become 'over-trained', causing them to function more like lookup tables than actual neural networks, making them unable to accurately classify any data that varied too far from the training set.

The results provided by these networks showed these initial concerns to be unfounded, displaying levels of accuracy equal to or greater than the average of the bracket networks. Despite these networks seldom having greater accuracy than every individual bracket network, the least accurate master network performed accurate classifications 89.6% of the time. A baseline accuracy of this degree, combined with the networks' high variance, make the master networks by far the best choice in terms of a neural network classifier.

6.4. Threshold Classification

In a system such as this, classification accuracy is only one half of the problem. The speed at which classifications can be performed is just as important, if not more important when ensuring that the system provides an efficient interface for any potential user. Dynamic thresholding was chosen to be the competitor to neural network classifiers for one reason alone: speed. While the networks required exhaustive distance calculations to build an accurate, rotation-free map of any stance before passing it through the series of weighted algorithms to finally produce confidence values for each stance, thresholding required just a few distances to be calculated on key points of the body and no external algorithms.

This, in theory, means that a thresholding classifier should always run faster than its neural network counterparts. The real question was whether or not the thresholding could keep up with the levels of accuracy provided by the networks. Unfortunately this was not the case - the

thresholding algorithms performed relatively well when classifying upper body stances, but were unable to differentiate between the lower body pair of stances 1 and 3, and the pair of stances 2 and 4. These pairs represented the left knee raise and left kick, and their right-side counterparts, respectively, and the thresholding classifier's inability to distinguish them from one another led to the lowest accuracy levels recorded for any classifier, with the lowest of these being 0.2%.

With accuracy results as low as this, dynamic thresholding cannot be considered as a viable classifier, irrespective of how much faster it may be than its neural network counterparts. The simplicity that gives the classifier its speed is also its greatest flaw, as each stance is classified as either true or false in each frame based on whether or not every one of that stance's component Booleans reads true. If a single component is false, the whole stance immediately reads as not being performed, which unfortunately does not offer anywhere near the same room for error that neural networks do. With this in mind, this binary nature also means that thresholding is the only type of classifier that can ever provide a reading of 100% accuracy, as shown for Stance 0 in Table 5 and Stance 3 Table 8.

6.5. Kinect Setups

Another factor to consider when examining the accuracy versus speed tradeoff is the Kinect setup responsible for capturing the data fed into the classifier. The dual Kinect setup added a great deal of redundancy when it came to the possibility of the occlusion issues that commonly plague optical capture, but the question was the cost incurred to the performance of the system. From an accuracy standpoint, it is plain to see that two Kinects are most assuredly better than one, with the dual setup outperforming its single counterpart in every average result aside from the solo upper body neural network.

It was initially assumed that classification performed on single Kinect captures would be far less accurate owing to occlusions invalidating large portions of data, but because those occlusions also took place in a large portion of the data used to train the neural networks, the networks learned to account for the invalid data and still classify those stances accurately. The same cannot be said for the thresholding classifier, though its issues stem from the subjective nature of optical

capture as a whole rather than the occlusion problem.

Unsurprisingly, the biggest difference between the results gathered from the single and double Kinect setups was not in their accuracy, but in the amount of time taken to perform the capture. On average, capture using two Kinects took roughly 25% longer than capture using only one, calling into question whether or not the small increase in accuracy provided by the second Kinect was worth the increase in capture time.

6.6. Framerate

Accurate classifications mean very little in a system that is unusable owing to a framerate throttled by the extensive lengths gone to when providing that accuracy. This is unfortunately the case across the board, with every single result scoring lower than the 90 FPS (Frames Per Second) required to provide an immersive VR experience. The thresholding classifiers performed exactly as intended on this front, nearly doubling the frame rate of their neural network counterparts, but still not breaking past 60 FPS, let alone 90.

Many factors contribute to the slow speed of most of these classifiers, the primary one being the need for external console applications. If Unity could have housed all of its Kinect and neural network interfacing, the system would not have had to spend nearly as much time on sending communications to other programs and waiting for responses before moving on to the next frame.

Neural networks are, by definition, extremely computationally intensive, so much so that they are trained using the computer's GPU rather than the CPU to save time more often than not. Despite the fact that the original idea of a single large, full-body, neural network with 190 input nodes was replaced by an upper and lower body pair with 55 and 36 input nodes, respectively, the use of the two small networks still requires too much in the way of the computer's resources to maintain an ideal framerate.

6.7. Stance Similarity

Up until this point, a final major factor affecting classification accuracy has been only been briefly touched on when discussing thresholding results: the similarity between multiple stances. Across the board, the least accurate results for every classifier can be found in the columns for stances 1 to 4 in the lower body tables, representing the left knee raise, right knee raise, left kick, and right kick stances. These provided a unique problem in that they were important inclusions for creating the feeling of earthbending, despite the fact that they were known to be very similar to one another while also physically being the hardest stances to perform.

Many of the subjects, in both the training and testing phases, showed that they lacked the flexibility in their legs to raise their knees as high as their hips, let along doing that with their leg extended for a kick. This lack of flexibility greatly reduced the variance between these stances for most of the subjects, which in turn made it harder to train the neural networks to differentiate between the pairs, and thus reduced the classifiers' ability to generate accurate confidences.

The final selection of ten stances was greatly narrowed down from an original pool of over 20 and, even then, the networks had great difficulty differentiating between some of them. Ten stances are unfortunately not nearly enough to properly flesh out a system designed to simulate earthbending, but the addition of even a single other stance will undoubtedly reduce classification accuracy across every neural network that incorporates it.

6.8. Classification Precision

The results in Tables 3-6 show the accuracy of classifications that had the greatest confidences out of the pool of output nodes in their respective neural networks; however, these results can be further refined. As mentioned previously when discussing how the system would be tested, classifications with over 90% confidence were recorded separately; the percentage of these 'precise' classifications are recorded in Tables 7-10.

As can be seen from the results in these tables, the vast majority of the classifications recorded throughout testing had confidences of over 90%. Tables 11-14 show what percentage of these

precise classifications were for the correct stance. The threshold row was not included in any of these tables as the binary nature of the thresholding algorithms means that their classifications are always precise. The average accuracy of the precision results for all three types of neural networks were greater than those of their broader counterparts in Tables 3-6. This shows that, by simply excluding classifications below a certain confidence percentage, it is possible to weed out many of the incorrect classifications produced by the networks.

If the confidence percentage had been pushed up to 95% instead, the system would no doubt have been able to exclude even more incorrect classifications. However, taking into consideration the level of variance trained into some of the networks with larger datasets, such as the mid and master networks, a 95% confidence threshold could prove to be too much. If environmental factors are also taken into account, reaching such a high threshold may not even be possible.

6.9. Practical Application of Results

Considering all of the results covered thus far, it is apparent that the system will never be viable for VR in its current state, but there are still a few key points that can be taken away from these tests. The first of these is how well the master networks performed. Considering that the system makes use of a branching structure to track progress through an action-triggering gesture by checking the order of stances performed, any user should be capable of triggering a desired action by performing certain stances in sequence. Even with the master network only capturing around 20 frames each second, each of those frames has, on average, over 90% chance of being classified correctly. If even one frame is classified correctly, the branching structure will advance closer to the leaf node that triggers the desired effect.

The second thing to take into account is that, while it definitely slowed the system down and was not as necessary for overcoming occlusions as originally thought, dual Kinect capture did increase the accuracy of the system. If the Kinect interfaces did not have to be run externally owing to compatibility issues with Unity, the detrimental effect it had on framerate could be greatly reduced.

The third and final consideration is the hardware specs of the computer used to run the testing. A Dell Inspiron 7567 laptop was used, housing an Nvidia GTX 1050ti GPU, an Intel Core 15-7300HQ CPU, 8GG DDR4 RAM at 2400MHz. GTX 1060 GPUs are widely considered to be the baseline GPU for running VR systems, immediately making this device subpar for testing purposes. A computer with higher specs was not used for testing because, while it would almost certainly have increased the framerate in every test, the argument could then be made that the system could have been tested on another computer with even higher specs, and so on.

7. Future Work

While the dual Kinect setup made substantial strides toward preventing occlusions as intended, the positional mismatch between the two Kinects' skeletons created a large amount of jitter in the visuals of the mean skeleton. In order to not have that jitter affect the user's experience in a manner harmful to the user's sense of presence, only a single Kinect should be used to represent the user. The backend of the system will maintain its dual Kinect capture and even make use of the second Kinect's data to fill in for the primary Kinect when occlusions occur, but the user will never see the mean skeleton being used to represent them during classification.

With the gradual release of new VR technologies, such as the Oculus Quest, systems that make use of the full 360 degrees of rotation are becoming increasingly more prevalent. The removal of cabling from HMDs because of the development of more powerful mobile hardware also provides the user with the opportunity to move much more freely than ever before, and could prompt the use of even more Kinect sensors in series, providing an even more accurate representation of the user by capturing them from four or more angles around the room. Running this would most certainly require a computer with higher specs than the one used during the recorded tests.

Continuing to explore examining the way stance data is captured, an entirely different capture medium could be used, such as utilizing a motion capture suit that houses various mechanical sensors used to track positional data across the wearer's body. Mechanical sensors completely remove the somewhat subjective nature and occlusion issues that plague optical capture, allowing for more accurate stance readings.

While re-examining stance capture, it is also extremely important to consider which stances are selected to be placed in the final pool. In order to create a complete gesture, stances should have a natural flow from one to the other, such as the stomp gesture performed by standing with both feet on the ground, raising one knee, and finally placing the raised leg back on the ground to resume the standing stance. It is also important to examine which stances are too similar to one another for the classifiers to accurately differentiate between. This can be achieved by starting with a large pool of stances, training a classifier to recognize all of them, and then testing extensively with that classifier to see which stances are most often being mistaken for one another or being classified with subpar accuracy. Those stances can then be eliminated one by one until the overall accuracy of the classifications if satisfactory while maintaining a suitably large pool of stances.

While increasing the number of stances a neural network is trained to recognize, increasing the number of subjects whose data it is trained on will further increase the accuracy of its classifications. This increase has diminishing returns and runs the risk of eventually turning the network into a glorified lookup table if it is trained too extensively. In order to avoid training a single network on too much data, it might be interesting to examine an alternative means for separating subjects into better batches, namely using standard deviation rather than a uniform three-way divide to create size brackets.

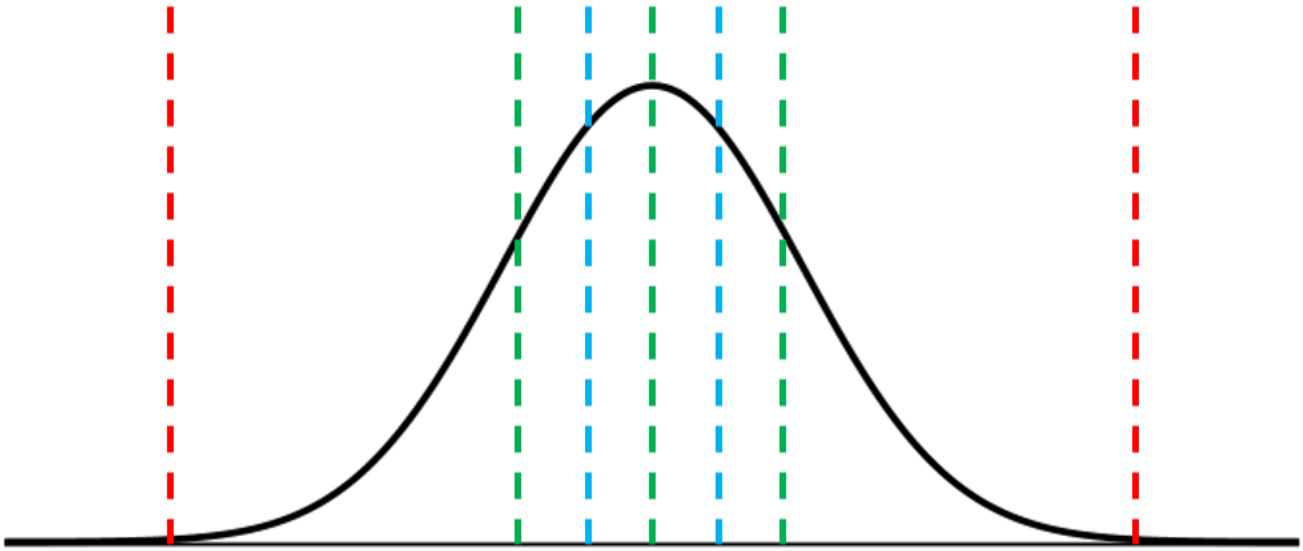


Figure 17 - Subject size distribution

As shown in Figure 17, the distribution of subjects based on their sizes forms a bell curve, where the curve ranges from short to tall on the X-axis and the number of subjects of a particular size is represented by the Y-axis. The central green line represents the mean size, while the left and right green lines represent the positive and negative standard deviation from that mean. Each of the areas between mean and standard deviation contains roughly 34% of the total number of subjects. The blue lines represent a move of half the standard deviation from the mean, and will become the new limits for what is considered 'average' size. This means of separation ensures that the average bracket still contains the most subjects while moving more into the short and tall brackets, increasing their variance.

In the dataset used in this study, most subjects are grouped tightly in the mid bracket, which is one of the reasons why the short and tall brackets perform so poorly. The subjects who fall just under or just over the limits for the mid bracket have sizes that are still more closely suited to the mid bracket than the bracket in which they are actually placed, thus skewing the results. If a standard deviation separator could be used instead, the current brackets would all contain a more uniform number of subjects, which would slightly lower the variance of the data in the mid bracket while greatly increasing the variance in the other two.

Touching on the increase in accuracy shown by focusing solely on classifications above a certain confidence level, the system could make use of a confidence threshold that scales down over time. This threshold could begin at as high as 99%, and slowly become more lenient with each new frame until the data supplied by one of those frames surpasses the threshold. When this happens, that stance classification would be returned to the system and the confidence threshold would revert to its initial value. Implementing such a technique would ensure that the system maintains high levels of classification accuracy while avoiding the prevention of classifications altogether in certain situations.

Another means of improving system performance that was not discussed is the idea of optimization. Memory leaks and poorly-written algorithms can slow down a system greatly, especially when that system is running many of these suboptimal processes every second. Something as simple as assigning a value in a multidimensional array a temporary variable of its own when it is bound to be used multiple times in a looping process can greatly reduce the time

for that process to complete its execution. A thorough examination and cleanup of the systems code could drastically speed up its execution.

Finally, testing should be performed on a high-end computer capable of running the system at a bare minimum of 90 FPS. This is the only true way to test if users actually feel more engaged when utilizing a full-body motion capture control scheme instead of their usual handheld controllers. It would be most beneficial to have a variety of computers with different combinations of hardware in order to see what individual change makes the most difference in terms of improving system performance.

8. Conclusions

The project set out with the intention of examining the feasibility of using a full-body motion capture control scheme in place of traditional controllers in an attempt to promote presence in a VR context. Of the various technologies examined, a combination of Kinect capture and neural network classification were chosen to provide the functionality needed to create this system. Many issues needed to be overcome along the way, including the Unity engine's rejection of external code libraries, animating an avatar in real-time using motion capture data, and breaks in data caused by optical occlusions.

Multiple training datasets were created from a large pool of capture subjects in order to determine which style of neural network would produce the most accurate classifications. Solo neural networks were initially assumed to be the best form of classifier, but turned out to have too little variance to account for changes in capture variables such as lighting and clothing. The size-bracketed networks were assumed to be the next best in terms of accuracy and, while they showed a significant improvement on the results of the solo networks, they failed to outclass the master networks.

Prior to testing, it was hypothesized that the master networks would show the worst results of all owing to their wide variance and the possibility of overtraining issues, but they showed the best results, on average, across all variations of testing. Disproving this initial assumption

increases the accessibility of the system, as it allows it to be trained long before it ever reaches users, making it much easier to pick up and use. If either of the other styles of networks had been more accurate, anyone using the system would have to first undergo a period of data capture in order to either train a solo network tailored to them, or to place them in a bracket suited to their size.

Unfortunately, the task of overcoming the optical occlusion and code library rejection issues took its toll on the system's performance. Understanding that this would become an issue, a simpler means of classification was created in the form of dynamic thresholding algorithms. For the most part, these proved to be more accurate at classifying stances than the solo neural networks, but were still far less accurate than the master networks. They did, however, show a significant increase in the speed at which the system could operate, though this rise in performance was still significantly below the 90 FPS baseline required to run a VR system.

The system that was developed proved to not be viable in its current state, largely owing to the constraints placed on it by the availability of hardware and Unity's lack of acceptance of the Kinect SDK and neural network NuGet packages. Should the system be run on a computer with hardware suitable to handle it, or should a better way to bridge the gap between Unity and external capture and classification be found, the system could very well become viable.

Despite the system not functioning at an appropriate speed, the techniques explored to create its gesture-based control scheme proved to be very effective. This success should aid in encouraging the creation of similar systems utilizing different variations and technologies to achieve similar goals, and push for the need for better solutions to the interoperability problems that held this system back.

9. Results Tables

Table 5 - Single Kinect Lower Body Overall Classification Accuracy Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Stance 4	Stance 5	Average Accuracy	Average Time (ms)	FPS
Solo	89.76925	51.31783	67.64567	52.32948	41.11297	43.85895	57.67235	8854.136	28.23539
Short	99.4924	70.60261	66.27258	70.0716	8.876685	99.38323	69.11652	8635.095	28.95162
Mid	99.29951	95.15499	92.97845	69.35455	82.20715	90.91881	88.31891	9076.835	27.54264
Tall	91.28983	82.90862	88.92339	64.89886	76.66127	98.69102	83.8955	9160.567	27.29089
Bracket	96.69391	82.88874	82.72482	68.10834	55.91504	96.33102	80.44364	8957.499	27.90958
Master	86.68314	79.81579	90.24729	76.35983	67.0724	89.60712	81.63093	8921.79	28.02128
Threshold	100	89.3913	77.53043	4.495652	0.2	81.8174	58.905797	4412.156	56.66164

Table 6 - Single Kinect Upper Body Overall Classification Accuracy Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Average Accuracy	Average Time (ms)	FPS
Solo	74.60439	56.35126	59.29044	70.2515	65.1244	10929.53	22.87381
Short	89.44437	89.8382	95.10172	99.41943	93.45093	9771.267	25.58522
Mid	92.01247	84.93116	89.82208	97.62873	91.09861	10310.81	24.2464
Tall	94.2711	92.09644	84.79414	98.30277	92.3661	10336.26	24.1867
Bracket	91.90931	88.95526	89.90598	98.45031	92.30521	10139.45	24.65618
Master	90.2305	89.84956	90.70209	98.08706	92.2173	10056.1	24.86054
Threshold	63.67868	69.57142	80.28486	99.14673	78.17042	4387.629	56.97838

Table 7 - Double Kinect Lower Body Overall Classification Accuracy Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Stance 4	Stance 5	Average Accuracy	Average Time (ms)	FPS
Solo	88.49004	78.5875	73.1125	80.77493	35.57296	75.90701	72.07416	11066.91	22.58986
Short	66.08538	56.66536	51.04188	95.71444	92.61692	93.89845	76.00374	10983.48	22.76147
Mid	98.4794	97.9276	99.27338	79.40474	80.479	86.19422	90.29306	11119.96	22.4821
Tall	99.29781	74.62705	97.85638	77.95063	70.7546	97.48885	86.32922	10700.92	23.36248
Bracket	87.9542	76.40667	82.72388	84.3566	81.28351	92.52717	84.20867	10934.78	22.86282
Master	89.1706	96.59355	95.28828	88.19172	78.44987	94.61154	90.38425	10975.13	22.77876
Threshold	99.50477	87.21905	91.19048	7.457143	2.476191	82.0381	61.64762	6705.275	37.28408

Table 8 - Double Kinect Upper Body Overall Classification Accuracy Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Average Accuracy	Average Time (ms)	FPS
Solo	74.16776	53.76054	55.64587	61.43075	61.25123	12193.97	20.50194
Short	99.31797	78.86501	97.83938	99.07853	93.77522	12085.54	20.68588
Mid	98.16054	83.98648	93.64078	99.40506	93.79821	12239.31	20.42599
Tall	97.2409	83.50871	91.08956	98.48215	92.58033	11789.23	21.2058
Bracket	98.23981	82.12007	94.18991	98.98858	93.3846	12038.02	20.76753
Master	96.41555	92.45936	91.9081	99.30804	95.02277	12071.22	20.71041
Threshold	92.9273	88.17023	89.23347	100	92.58275	6730.697	37.14325

Table 9 - Single Kinect Lower Body Precision Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Stance 4	Stance 5	Average Precision
Solo	97	90	69	79	69	68	78.67
Short	100	58	78	65	70	100	78.5
Mid	93	90	92	79	56	94	84
Tall	95	95	80	56	77	98	83.5
Bracket	96	81	83.33	66.67	67.67	97.33	82
Master	91	87	83	67	52	90	78.33

Table 10 - Single Kinect Upper Body Precision Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Average Precision
Solo	95	76	67	89	81.75
Short	93	95	98	100	96.5
Mid	95	98	93	99	96.25
Tall	96	77	85	99	89.25
Bracket	94.67	90	92	99.33	94
Master	92	98	96	98	96

Table 11 - Double Kinect Lower Body Precision Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Stance 4	Stance 5	Average Precision
Solo	89	88	87	62	65	95	81
Short	63	50	60	81	55	99	68
Mid	91	88	95	55	45	83	76.17
Tall	100	62	88	80	54	89	78.83
Bracket	84.67	66.67	81	72	51.33	90.33	74.33
Master	94	86	88	65	57	91	80.17

Table 12 - Double Kinect Upper Body Precision Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Average Precision
Solo	91	89	98	97	93.75
Short	99	88	96	100	95.75
Mid	99	97	99	100	98.75
Tall	99	94	99	99	97.75
Bracket	99	93	98	99.67	97.42
Master	97	96	98	99	97.5

Table 13 - Single Kinect Lower Body Precise Classification Accuracy Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Stance 4	Stance 5	Average Accuracy
Solo	94	66	62	62	56	48	64.67
Short	100	97	98	72	14	99	80
Mid	99	98	96	80	94	97	94
Tall	97	93	92	65	92	99	89.67
Bracket	98.67	96	95.33	72.33	66.67	98.33	87.89
Master	96	89	93	83	75	92	88

Table 14 - Single Kinect Upper Body Precise Classification Accuracy Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Average Accuracy
Solo	76	54	60	77	66.75
Short	94	90	98	99	95.25
Mid	94	88	90	98	92.5
Tall	95	92	88	98	93.25
Bracket	94.33	90	92	98.33	93.67
Master	92	92	93	99	94

Table 15 - Double Kinect Lower Body Precise Classification Accuracy Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Stance 4	Stance 5	Average Accuracy
Solo	92	88	86	90	42	87	80.83
Short	99	88	53	97	97	99	88.83
Mid	98	99	99	71	93	97	92.83
Tall	99	75	98	83	71	97	87.17
Bracket	98.67	87.33	83.33	83.67	87	97.67	89.61
Master	99	97	96	92	92	98	95.67

Table 16 - Double Kinect Upper Body Precise Classification Accuracy Results

Classifier	Stance 0	Stance 1	Stance 2	Stance 3	Average Accuracy
Solo	76	56	61	65	64.5
Short	99	92	98	99	97
Mid	98	88	96	99	95.25
Tall	99	82	90	99	92.5
Bracket	98.67	87.33	94.67	99	94.92
Master	97	93	95	99	96

10. References

Beat Games. *Beat Saber.* 21 May 2019. Web site. 10 October 2019.

Berger, Kai, et al. "Markerless Motion Capture using multiple Color-Depth." *Vision, Modelling and Visuallisation* (2011). Journal Article.

—. "Markerless Motion Capture using multiple Color-Depth Sensors." NA: The Eurographics Association, 2011.

Brunner, Grant. *Geek Deals: Save Big on Oculus Rift and HTC Vive*. 24 August 2017. Web site. 15 July 2018.

Changhau, Isaac. *Loss Functions in Neural Networks*. 7 June 2017. Web site. 5 July 2018.

Cong, Robert and Ryan Winter. *How Does The Xbox Kinect Work*. NA. 23 April 2018. <https://www.jameco.com/jameco/workshop/howitworks/xboxkinect.html>.

Daljo. *Avatar the Last Airbender – Fire Scroll Poster.* n.d. Web site. 12 October 2019

Dujmovic, Jurica. *Here's why you will be hearing more about virtual reality*. 15 July 2019. Article. 2 October 2020.

Endnight Games Ltd. *The Forest.* 30 April 2018. Web site. 10 October 2019.

Guerra-Filho, Gutemberg B. "Optical Motion Capture: Theory and Implementation." *SIBGRAPI* (2005). Journal Article.

Hardawar, Devindra. *Xsens body suits are getting even better at motion capture*. 17 June 2017. Web site. 15 july 2018.

IEEE VGTC. *IEEE VGTC: Virtual and Augmented Reality Technical Awards*. 2015. 23 April 2018. <http://vgtc.org/content/virtual-and-augmented-reality-technical-awards>.

JaketheSnake486. *Martial Arts Influences in Avatar: The Last Airbender*. 5 October 2014. Web site. 15 July 2018.

Janocha, Katarzyna and Wojciech Marian Czarnecki. "On Loss Functions for Deep Neural Networks." *Theoretical Foundations of Machine Learning*. Kraków, 2017. Coference Paper.

Lamkin, Paul. *Best VR headsets 2018: HTC Vive, Oculus, PlayStation VR compared*. 2018. 22 April 2018. <https://www.wareable.com/vr/best-vr-headsets-2017>.

Leon, Keno. *Making a Simple Neural Network : Classification*. 2017. 23 April 2018. <https://becominghuman.ai/making-a-simple-neural-network-classification-2449da88c77e>.

Lonkar, Sagnar. *Types of Motion Capture*. n.d. Web site. 3 July 2018.

Microsoft Kinect Webcam. n.d. Web site. 15 July 2018.

Moawad, Assaad. *Neural networks and back-propagation explained in a simple way*. 1 February 2018. Web Site. 5 July 2018.

"Motion Capture: Magnetic Systems." *Next Generation* October 1995. Magazine Article.

Murray, Janet. *Hamlet on the Holodeck: The Future of Narrative in Cyberspace*. New York: The Free Press New York, 1997. Book.

Nielsen, Michael A. *Neural Networks and Deep Learning*. Determination Press, 2015. Book.

Optical Motion Capture Systems. n.d. Web site. 5 July 2018.

Orland, Kyle. *Microsoft Announces Windows Kinect SDK For Spring Release*. 21 February 2011. Web site. 5 July 2018.

Perz, Małgorzata. "Flicker perception in the periphery." 13 May 2010. *IEIS.* Document. 23 August 2018.

Pham, Alex. "E3: Microsoft shows off gesture control technology for Xbox 360." *Los Angeles Times* 1 June 2009. Article.

Raiten, Shai. *Kinect – Getting Started – Become The Incredible Hulk*. 17 June 2011. Web site. 15 July 2018.

Roetenberg, Daniel, Henk Luinge and Per Slycke. "Xsens MVN: Full 6DOF Human Motion TrackingUsing Miniature Inertial Sensors." *Xsens Technologies* 3 April 2013. Article.

Rousseau, Robert. "A History and Style Guide of Hung Gar Kung Fu." 9 August 2017. *ThoughtCo.* Article. 24 August 2018.

Schepers, Martin, Matteo Guiberti and Giovanni Bellusci. "Xsens MVN: Consistent Tracking of Human Motion Using Inertial Sensing." Technical Report. 2018. Document.

Schwartz, Alex and Devin Reimer. "The Holodeck is here - Designing for Room-Scale VR." Boston: Unite, 2015.

Sharma, Sagar. *Activation Functions: Neural Networks*. 6 September 2017a. Web site. 5 July 2018.

Sharma, Sagar. *Epoch vs Batch Size vs Iterations*. 23 September 2017b. Article. 2 October 2020.

Slater, Mel and Sylvia Wilbur. "A framework for immersive virtual environments five." *Teleoperators and Virtual Environments* 6.6 (1997): 603-616.

Totilo, Stephen. *Microsoft: Project Natal Can Support Multiple Players, See Fingers*. 6 May 2009. Article. 5 July 2018.

Wing Lam Kung Fu. "Character of Hung Gar in Combat." 1 September 2011. *Wing Lam Kung Fu.* Article. 24 August 2019.

wrnchAI. n.d. Web site. 5 July 2018.

11. List of Figures

Figure 1 – Beat Saber (Beat Games)

Figure 2 – The Forest (Endnight Games Ltd)

Figure 3 – Oculus Rift and HTC Vive HMDs (Brunner, 2017)

Figure 4 – Xsens motion tracking suit and its digital mappings (Hardawar, 2017)

Figure 5 – First generation Kinect sensor (Microsoft Kinect Webcam)

Figure 6 – Architecture of a neural network (Nielsen, 2015)

Figure 7 – Firebending examples (Dalio)

Figure 8 – Earthbending examples (JaketheSnake486, 2014)

Figure 9 – Kinect skeletons (Raiten, 2011)

Figure 10 – Kinect skeleton height comparison

Figure 11 – Node addition in neural networks

Figure 12 – Single vs dual Kinect capture to address occlusion issues

Figure 13 – Kinect cubeman skeleton

Figure 14 – Rock golem used for body matching

Figure 15 – Neural network training metrics before applying the zero fix

Figure 16 – Neural network training metrics after applying the zero fix

Figure 17 – Subject size distribution

12. List of Tables

Table 1 – Loss functions (Janocha and Czarnecki, 2017)

Table 2 – Activation functions (Sharma, 2017a)

Table 3 – Relative Body Measurements

Table 4 – Bracket Thresholds

Table 5 – Single Kinect Lower Body Overall Classification Accuracy Results

Table 6 – Single Kinect Upper Body Overall Classification Accuracy Results

Table 7 – Double Kinect Lower Body Overall Classification Accuracy Results

Table 8 – Double Kinect Upper Body Overall Classification Accuracy Results

Table 9 – Single Kinect Lower Body Precision Results

Table 10 – Single Kinect Upper Body Precision Results

Table 11 – Double Kinect Lower Body Precision Results

Table 12 – Double Kinect Upper Body Precision Results

Table 13 – Single Kinect Lower Body Precise Classification Accuracy Results

Table 14 – Single Kinect Upper Body Precise Classification Accuracy Results

Table 15 – Double Kinect Lower Body Precise Classification Accuracy Results

Table 16 – Double Kinect Upper Body Precise Classification Accuracy Results