Surrogate Robot Body V1: Immersive Telepresence System

by: Prakash Waghela

Student ID: B221256

Supervisors:

Dr Firat Batmaz

Dr Andrea Soltoggio

16COC251: Computer Science Project

Department of Computer Science

Loughborough University

May 2017

Acknowledgements

I would like to thank Dr Firat Batmaz for his supervision for the duration of the project. His guidance and insights were helpful in completing the project.

I would like to thank Dr Andrea Soltoggio for organising access to the equipment required to conduct this project and for helping to communicate with other students that were undertaking similar projects.

Finally, I would like to thank Puja Ramji, a relative who sadly passed away this year. You provided a mountain of support in life and you continue to inspire me in death.

Abstract

This project aims to explore how the collaboration of individual pieces of hardware can be used to create an advanced telepresence system. Four devices; a Myo Armband, Leap Motion, Oculus Rift DK2 and a Nao robot, have been reviewed and the most appropriate of the four have been selected to be utilised to build a telepresence system. Upon selecting the most appropriate devices, I have attempted to take advantage of the strengths of each device in my project deliverable to develop a control scheme for the Nao robot. The control scheme aims to allow the user to maintain a high level of control over the Nao robot; enough that the robot is able to interact with the environment in a meaningful way.

Table of Contents

Chapter 1 - - Introduction

1.1 Project Summary

This project involves the collaboration of multiple devices to create a functional telepresence system.

The challenges in this project include:

- Reviewing the variety of technology available for the project and deciding what would be suitable for the project.
- Learning how each device can be utilised and how to take advantage of the unique benefits each device provides.
- Working with hardware that may not be completely reliable and consistent.

Despite completing the project, I have limited the scope of the project to target the functionality used to control the robot. In implementing and focussing on the control scheme used for the project, I hope to leave the project open to improvement in subsequent years.

1.2 Literature Review

In this section, I have explored concepts connected to the scope of this project, looking at relevant related work and discussing the significance in relation to my project.

Telepresence can be defined at a basic level as the long-distance projection of oneself, with “tele” meaning long distance, and presence representing the projection of oneself (Cambridge Dictionary (a), 2013) (Cambridge Dictionary (b), 2013) . Under this definition, as a human race we have been capable of telepresence since the first telephones were invented. Under the correct pretences, something as simple as an email could be regarded as telepresence.

However, a more modern interpretation of the word telepresence would usually include some tangible, physical body that allows the user of the telepresence system to physically interact with the environment around them, as if they were there. Such an experience is impossible through phones or emails as the amount of feedback available through such mediums pales in comparison to the amount of feedback available by being there in person. Thus, one method to incorporate a more sophisticated telepresence system is to incorporate a medium that will allow for more feedback about the environment to be fed back to the user. The telepresence industry has recognised the field of robotics as a possible avenue to pursue to allow for this.

However, telepresence with robotics is not a new concept in the technological field, with efforts at humanoid robotics for telepresence dating back through the last thirty years. An example, the P2 developed by Honda in 1996, was capable of walking and boasted 34 degrees of freedom (Popović, 2013). Since then, many efforts have been made to improve upon this concept. Many devices have been released commercially, with a common use case being to assist those with mobility issues. In such a case, the control scheme of the system must allow for the user to be able to control the mobility of the robot without leaving any strain on the user.

The use of a gesture control scheme for a telepresence system could be detrimental for a patient that struggles with mobility. However, there have been studies which indicate that the use of gesture control could be used for rehabilitation purposes. One such study (Deutsch, Borbely, Filler, & Karen, 2008) investigated how games which implement a gesture control scheme could be used in the rehabilitation of an adolescent with cerebral palsy. The study found that the patient improved their posture control and functional mobility through activities that included games which implemented gesture control schemes.

This shows that there could be benefits to implementing a gesture control scheme to a telepresence system. However, the games used in the study make use of very simple control schemes. This is important to prevent the user from requiring a large amount of training to use the system. Also, by increasing the complexity of the control scheme, there is a possibility of reducing the reliability of the control scheme, thus the control scheme should be kept relatively simple.

1.3 Technical Review

1.3.1 Myo Armband

The Myo Armband (Thalmic Labs (a), 2017) is a wearable band that can be worn around the forearm of the user.

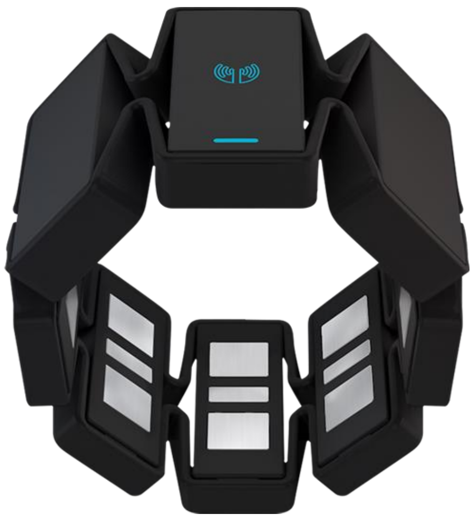


Figure 1 - A Myo Armband (Thalmic Labs, 2016)

Sensors

- 8 Electromyography sensors
    - The EMG sensors can be used to detect the electrical signals in the muscle. These readings can be manipulated to detect specific hand gestures and motions.
    - The raw EMG readings can be captured from the Myo through the SDK with a single reading being available per sensor, at a rate of 200Hz (Bernhardt, 2015).
- Three-axis gyroscope
    - The gyroscope can be used to detect rotational movement of the arm. There are three planes in which an object can be rotated (roll, pitch, yaw), all of which can be measured by the sensor.
    - The data from the gyroscope can be captured through the SDK, with a single reading in degrees per second (Thalmic Labs (b), 2017) being available for each rotation axis at a rate of 50Hz (Bernhardt, 2015).
- Three-axis accelerometer
    - The accelerometer can be used to detect movement of the arm. In specific, the sensor can detect the change in velocity and position of the arm.
    - The data from the accelerometer can be captured from the Myo through the SDK, with a single reading in units of G (Thalmic Labs (b), 2017) being available per sensor at a rate of 50Hz (Bernhardt, 2015).

- Three-axis magnetometer
  - A magnetometer can be used to detect magnetic fields. This allows allow the armband to act as a compass, using the Earth's magnetic field to determine the absolute orientation.

Interface

- Connectivity - The Myo armband has a Bluetooth interface that allows the user to connect the armband to a variety of compatible devices. This means the device can be operated without the use of wires, allowing for unrestricted movement while using the device.
- Battery - The Myo armband has a built-in battery which can be recharged through a micro USB port on the device. The battery is advertised to allow for "One full day use out of a single charge" (Thalmic Labs (a), 2017).
- Feedback - The Myo armband can provide direct feedback through two methods.
  - Haptic - The armband utilises short, medium, and long vibrations to notify the user of different events.
  - LEDs - The armband utilises LEDs on the device to notify the user of different device statuses.
- SDKs - Native SDK is available in C++.

Usability

The Myo armband is a lightweight and small device which easily fits around the forearm of the user.

- Gestures - The software for the Myo armband has five pre-programmed hand gestures (Thalmic Labs (a), 2017) that it can detect through the EMG sensors.
- Calibration - The software runs the user through a calibration step to create a user profile for the armband. This improves the reliability of the device and is specific to each user due to the variation in EMG readings through the arm of the user.

1.3.2 Leap Motion

The Leap Motion is a small, bar-shaped motion detector. The device can be placed on a flat surface and comes packaged with software to detect and track the user's hands.

Figure 2 - A Leap Motion Device (Leap Motion Inc, 2015)

Sensors

- Infrared Cameras - All data provided by the Leap Motion is derived from images taken by two infrared cameras contained within the base (Colgan, 2014), taken at up to 200 frames per second (Leap Motion Inc (a), 2017). The Leap software analyses the images and provides data from the motion of the tracked hands.
    - The cameras utilise wide angle lenses creating a large interaction space which is shaped like an inverted pyramid (Colgan, 2014).
    - The images are fed into the Leap Motion software which returns a 3D representation of the device’s field of view. The software can detect when a user’s hand is captured in an image and the software infers a large amount of data from the analysis of the image, allowing for the user’s hand to be tracked in 3D space.

Interface

- Connectivity - A single wire is required to connect the Leap Motion device to any USB port on a compatible device. The device is powered through the USB connection and does not require charging.
- Feedback - The Leap Motion has a green LED on the side of the panel to indicate that it is connected to a PC. When powered and connected, there are three red lights lit on the surface of the panel. All other feedback is given through the software.
- SDKs - Native SDKs are available in a wide variety of languages, including C++, Python, JavaScript, Java, and C#.

Usability

Once set on a surface, the Leap Motion is easy to control through the supplied software.

- Visualizer - Software to build a visualisation of any hands detected over the Leap Motion in real-time comes packaged with the software. This visualizer is useful for quickly testing whether a given hand motion is captured correctly by the device.
- Troubleshooting - Within software package is a troubleshooting page which allows the user to see any faults detected in the hardware at a glance.

1.3.3 Oculus Rift (DK2)

The Oculus Rift DK2 is a head mounted virtual reality headset. The device is a developer’s kit and as such has lower specs than the consumer device that is available on the market.

Figure 3 - An Oculus Rift DK2 (Hutchinson, 2014)

Sensors

- The headset contains a gyroscope, accelerometer, and magnetometer (Oculus, 2017). The data from these sensors is combined with a model of a user's neck and head to translate the real-time head movements made by the user into rotational data (pitch, yaw, roll) which can be sampled at up to 1000Hz (Oculus Blog, 2013).
- The headset contains an array of infrared LEDs which are tracked by the external camera supplied with the device (Oculus, 2017). The camera is used to track the LEDs, and as such track any movements in 3D space. This measurement can be sampled at 60Hz (VR&AR Wiki, 2017) (RiftInfo.com, 2016).
- The Leap Motion comes with an add-on mount which allows the user to slot the Leap Motion device onto the front of the Oculus Rift. The Leap Motion can then be utilised by tracking the user's hand in front of them rather than above the Leap Motion on a surface.

Output

- The headset contains two lenses, with each lens able to display media with a resolution up to 960 x 1080 at a refresh rate of 75Hz (VR&AR Wiki, 2017) (RiftInfo.com, 2016). Due to the high graphic requirements, a powerful PC is required to utilise the output capabilities of the Oculus Rift.
    - Any image that is to be output using the Oculus Rift must be pre-processed to account for the distortion required for the image to be seen correctly.

Interface

- Connectivity - Utilises multiple cables and requires a high-performance PC to function.
    - Could possibly utilise a lower-performance PC if only the positional tracking was utilised as the graphics rendering is the cause of the high-performance PC requirement.
- SDK - Native SDK is available in C++.

Usability

The Oculus Rift is a device which requires you to be tethered to the PC with multiple wires.

- The headset requires a tight fit to create an immersive experience. This can be uncomfortable for some users.
- To utilise the tracking capabilities, you must remain within the view of the camera used to track 3D motion.
- Due to the multiple components and wires required to correctly operate the device, there are sometimes sync issues with one component not being recognised correctly.
- The Oculus Rift DK2 requires a powerful PC with the following minimum specifications:
    - NVIDIA GTX 970
    - Intel i5-4590 or better
    - 8GB+ RAM
    - 1x HDMI, 3x USB (3.0)

1.3.4 Nao Robot

The Nao robot is a humanoid robot that can be programmed to carry out a wide variety of tasks (Aldebaran Documentation (a), 2016).

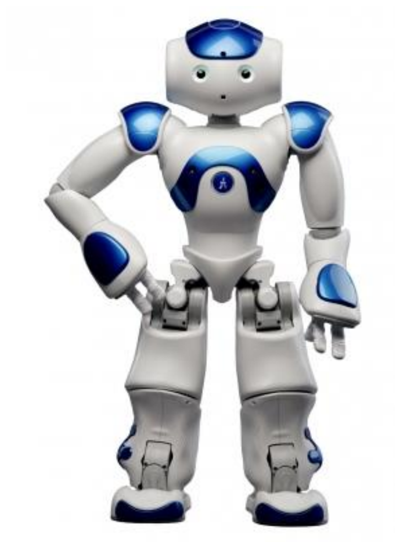


Figure 4 - The Nao Robot (Aldebaran - Softbank Robotics, 2014)

Sensors

- Capacitive tactile sensors - The robot has tactile sensors installed on its head, hands, and feet. Along with these sensors is a button located on the chest.
- Gyroscope and Accelerometer - The robot has access to inertial data, centred around the robot's torso (Aldebaran Documentation (a), 2016).
- Sonar - The robot has sonar capabilities that allow it to estimate the distance from objects ahead.
- Microphone - The robot has microphones installed on its head that provide a method for sound to be captured.

Output

- Loudspeakers - The robot has speakers installed allowing it to broadcast sound.
- Video Camera - The robot has two cameras installed which is capable (in ideal conditions) of providing video of resolution up to 1280 x 960 at 30 frames per second (Aldebaran Documentation (b), 2016).
    - In real conditions, the bandwidth of the network connection limits the camera's performance to a much lower resolution and frame rate.
- LEDs - The robot has a multitude of LEDs installed which are used to signify different robot states. The predominate LEDs are located on the head, in the eyes, ears, and within the chest button.
- Motion - The robot has 5 parts that make up the body of the robot, which utilises a total of 25 separate motors to allow for a wide array of movement.

Interface

- Connectivity - Connection to the robot is possible via an access point. The robot has support for either a wired ethernet connection or a wireless Wi-Fi connection.
- Choregraphe - A software suite available from the Aldebaran website which has an interface to view and broker connection details with the robot and sample movements through a drag and drop interface.
- Virtual Robot - A virtual version of the robot with reduced functionality is made available through Choregraphe.

- Open Nao - The firmware installed on the robot which is based on a Linux environment. The local environment can be accessed through an SSH client.
- SDKs - Native SDKs are available in C++, Python, and Java.

Chapter 2 - Solution Design

2.1 Aims and Objectives

The over-arching aim of this project is to create an interface which can be utilised as a telepresence system. This aim can be split into the following objectives:

2.1.1 Functional Objectives

1. The user will be able to control the movement of the robot in 3D space, using some gesture as the input.
2. The user will be able to control the movement of the head of the robot, using some gesture as the input.
3. The user will be able to control each arm of the robot individually. The arm should be able to be controlled to a level where the robot is able to interact with the environment.
4. The user will be able to utilise the speakers on the robot to relay some message.
5. The user should be able to get feedback from the robot, in the form of a camera image.
6. The user should be able to control the robot while it's untethered.
7. The motions required to move the robot will be relatable to the motion made by the robot.
8. The movement of the robot should be controlled and measured; any erratic movements or latency should be minimised.

2.1.2 System Objectives

1. The delivered project should be usable by any person with minimal training.
2. The delivered project should run on either Windows, Mac, or Ubuntu.
3. The delivered project may require a PC capable of utilising the Oculus Rift hardware.

2.2 Project Plan

After reviewing the technology available for this project, I decided to split work into three distinct phases. The purpose of these three phases was to ease the amount of work required at the end of the project, in phase three.

This was made possible as the main functionality in the project deliverable could be split up into a, communicating with the input devices, and b, communicating with the output devices. Thus, if the work to broker the individual device communication could be completed separately, the leftover work required to complete the project in phase three would be brokering the inter-device communication.

I decided on a programming language based on my initial research on the devices. As I would be combining work from the separate phases, I decided that using a language that was supported by each of the devices would be best, thus I chose to use C++. This decision was supported by looking at the previous attempt at this project, in which the project deliverable was written in C++.

2.2.1 Phase One - Input Devices

In the first phase of work, I planned to examine the methods of gathering input gestures from the user. The three devices I planned to investigate were the Leap Motion, the Myo Armband and the Oculus Rift.

The aim of this phase was to gain an understanding of how each input device functions, gain experience working with each device independently, and build up an idea of how each device could best be utilised in the deliverable of the project.

I planned to gain this experience by researching each device and writing a simple, console-based prototype for each device that would output the raw inertial data gathered by the device. I would then analyse the raw data to check the clarity of the data collected, allowing me to draw some conclusion on the fitness of the device.

2.2.2 Phase Two - Output Devices

In the second phase of work, I would examine the devices that could be used to give some output to the user. The two devices I would investigate were the Oculus Rift and the Nao Robot.

Like phase one, the aim of this phase was to gain an understanding of how each output device functions, gain experience working with each device, and build up an idea of how each device could be used in the project deliverable.

I planned to gain this experience by sending simple output signals to each device. For the Oculus I planned to research and test methods of rendering images to the lenses. For the Nao Robot, I planned to write a console-based prototype that would allow the user to have some simple form of control over the robot using a standard computer keyboard, using a method that could be easily replicated with the input devices.

2.2.3 Phase Three - Project Deliverable

In the third and last phase, I would look at combining work from the first two phases to create the final project deliverable.

It was difficult to plan the lower level details for the final phase before the completion of the first two phases due to the dynamic nature of the project. The content of the project deliverable depended entirely on the conclusions drawn from the first two phases. Thus, the first two phases were completed keeping the last phase in mind. This ensured that some design and programming work necessary for the project deliverable would be completed in conjunction with phase one and two.

2.3 Gantt chart

The Gantt chart below shows the timescale of each phase.

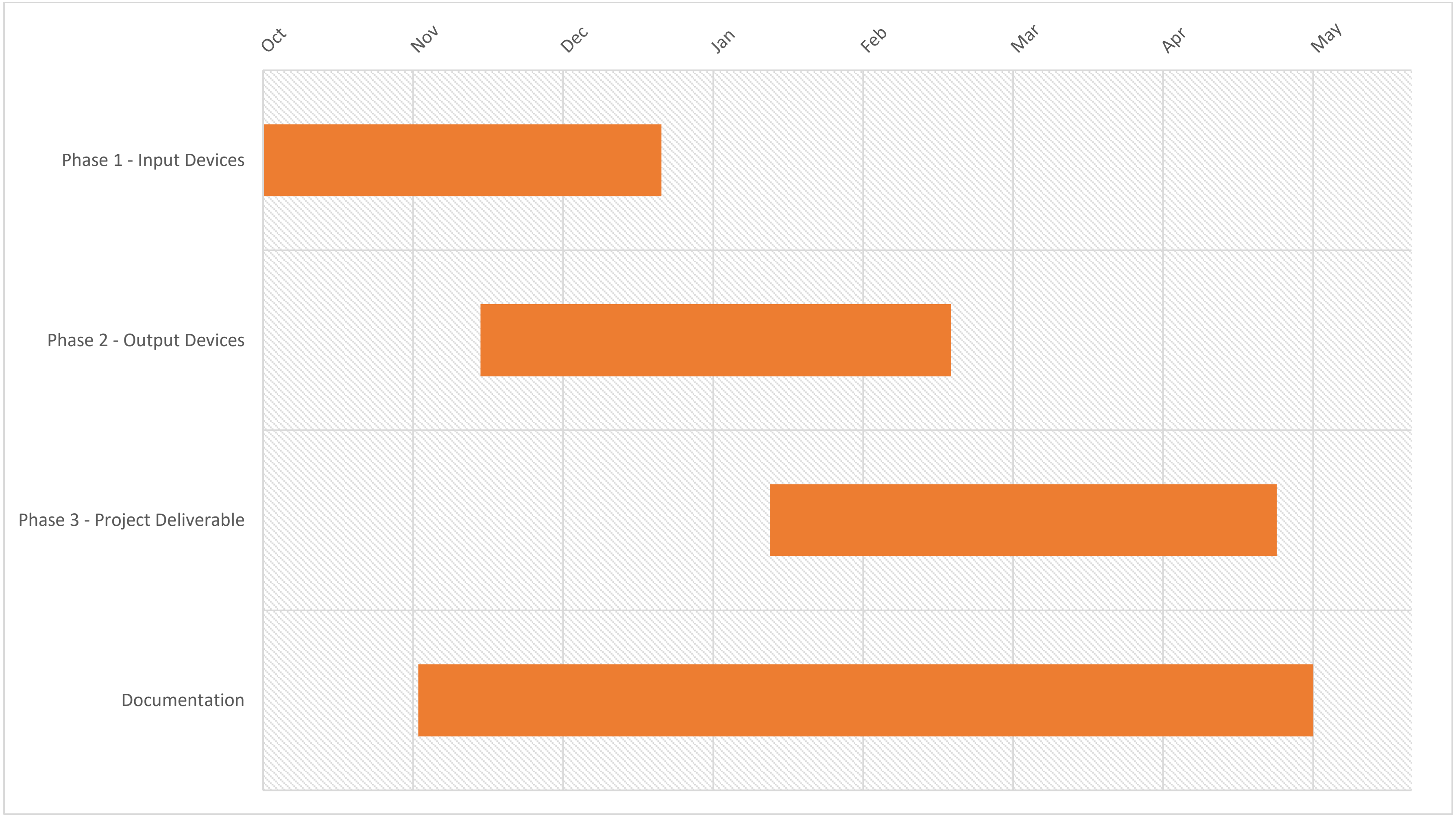


Figure 5 - A Gantt chart showing the timeline for the project

Chapter 3 - Implementation

3.1 Phase 1 - Input Devices

In the first implementation phase, I wanted to investigate the capabilities of the input devices. I decided to make a general assessment of each device on the following criteria:

- Availability of data
- Availability of documentation and support
- Reliability of data
- Ease of use
- Drawbacks

3.1.1 Myo Armband

I found that the Myo Armband was a lightweight device that was comfortable to wear and very portable. The device ran on battery power but I found that the device did not have any stand-by mode. Thus, the device required charging before each use as any leftover charge from the previous use would have been wasted while the device was idle.

I found that working with the armband was very easy. The software package needed to broker the connection with the armband installed without any problems and I was up and running very quickly.

I completed a tutorial on using the armband as part of the software package which directed me through the hand gestures that the armband could recognise.

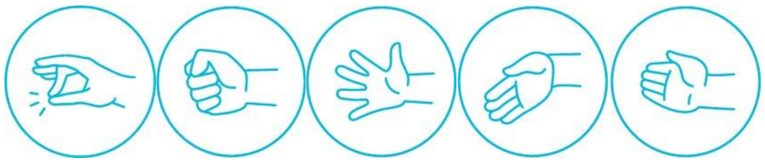


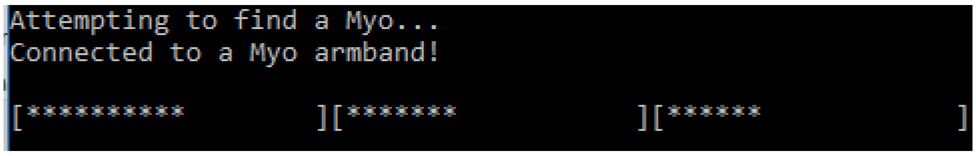
Figure 6 - The pre-programmed hand gestures for the Myo Armband (Thalmic Labs, 2016)

However, I found that the pre-programmed gestures were not reliably recognised by the armband. During the tutorial, a specific gesture was required to move onto the next stage of the tutorial but there were times the required gesture would not be recognised. I found that it was possible to create a “user profile” which mapped my real gestures to the pre-programmed gestures.

However, I found that while this improved the gesture recognition, some gestures required exaggerated movements to be detected correctly. For example, for the “fist” gesture to be correctly detected, I would need to ball my hand into a tight fist and tense my arm for the gesture to be detected, rather than balled my hand into a relaxed fist like I would prefer.

Upon completion of the tutorial, I moved on to sampling some applications available from the Myo application launcher. One such application allowed me to type on an on-screen keyboard, using my finger to point at letters and a gesture to select it. I found that I was able to point at individual letters very accurately but I could not select letters that easily. This indicated that the motion tracking aspect of the Myo worked well but the gesture tracking aspect wasn’t very reliable, which was supported by my experience with the tutorial.

Following this, I moved onto programming for the armband, I found that the C++ SDK had sufficient documentation and was bundled with some sample programs that demonstrated how to connect to the device and pull raw data from the sensors.



```
Attempting to find a Myo...
Connected to a Myo armband!

[**********          ][*******           ][******                ]
```

Figure 7 - Sample Application

As shown in Figure 7, the sample application had a visualisation of the inertial data available from the device. I checked the code to see if any pre-processing had been applied to the data and found that the visualisation was using the raw data. Using this sample application, I was able to see that the gyroscope in the Myo worked well; the data produced was very responsive to my movements and provided very accurate readings.

Following this, I wanted to experiment with the EMG sensors to gain some insight on why the gesture detection wasn't working very well. I modified the sample program to record the raw EMG readings from the armband and save them to a file. I ran the script whilst keeping my arm stationary and imported the file produced to an Excel spreadsheet, averaging, and plotting the readings from each individual sensor onto a line graph.

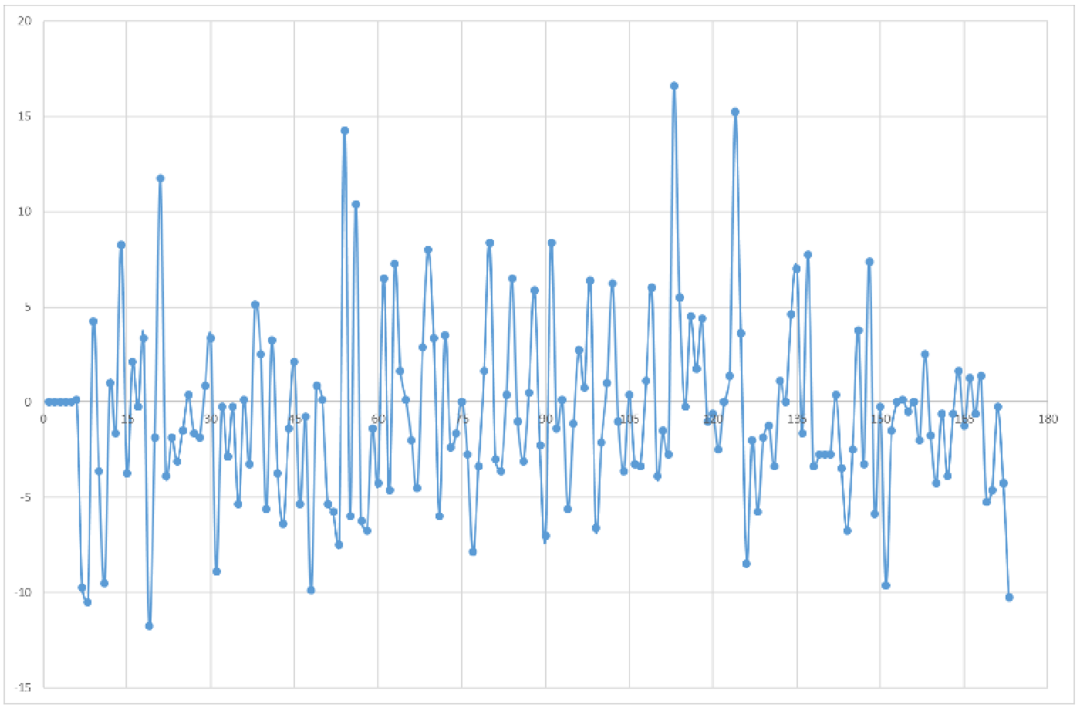


Figure 8 - Graph of EMG readings

Despite the low level of analysis, I expected to see some form of a pattern emerging in the data. However, as shown in Figure 8, the data proved to be very noisy. Despite the potential for some meaningful finding from the EMG readings through deeper analysis, I decided that the EMG readings contained too much noise to be able to accurately capture the movement of the arm.

3.1.2 Leap Motion

Like the Myo, I found that the Leap Motion was easy to set up. The software installed smoothly and connected with the Leap Motion device easily.

As mentioned in the technical review, a visualizer was bundled with the installed software. This visualizer would draw a wireframe hand and trace the movement of your hand as detected by the device. I used this visualizer to get a feel for how the device could cope with different hand motions.

I found that the device required a clear, unobstructed view of your hand for the tracking to be effective. There were some cases where some fingers were incorrectly displayed, normally occurring when some fingers were obscured from view by other fingers. In Figure 9, only my middle finger was perpendicular to my palm but the visualizer incorrectly detected two fingers being perpendicular.

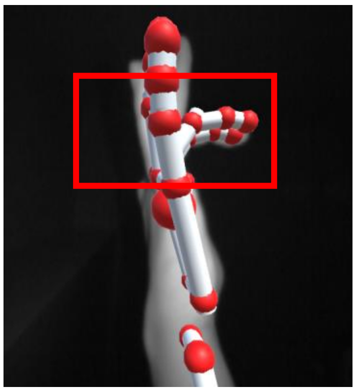
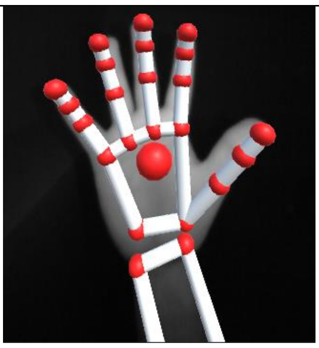
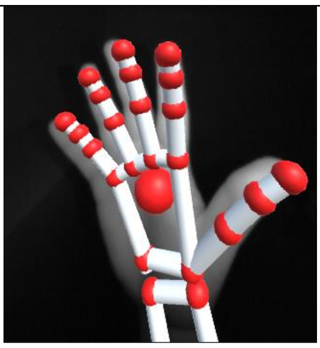
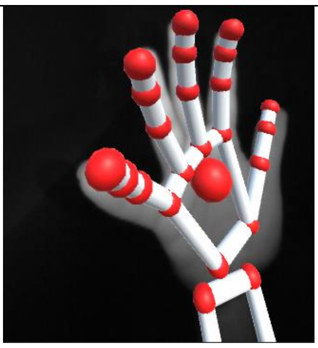
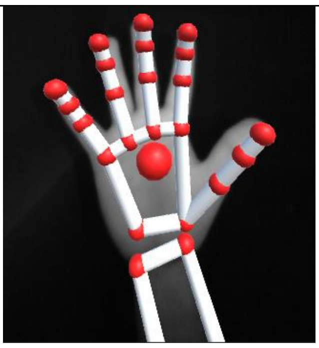
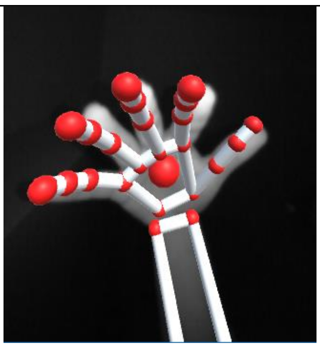
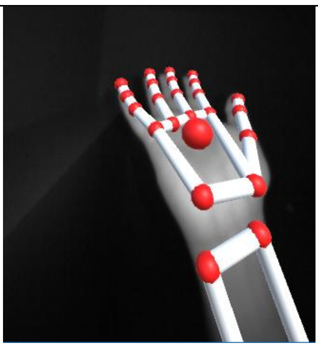


Figure 9 - Leap Motion visualizer showing an incorrect representation

I found that the device could track the position and rotations of my palm well. To test this I started with my palm facing down and rotated my hand about each axis, watching the visualisation to see how well the software could track my hand. Results of this test are shown in Table 1.

Axis	Normal	Anticlockwise	Clockwise
Roll			
Pitch			

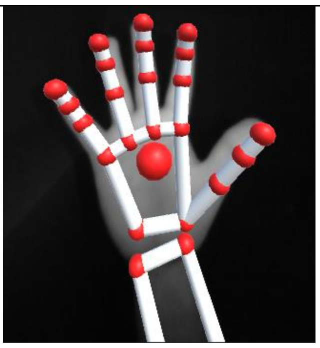
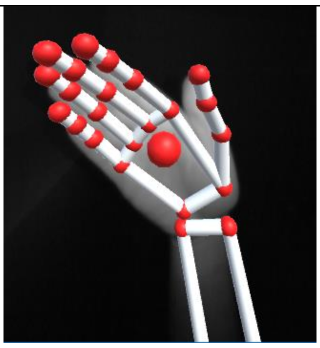
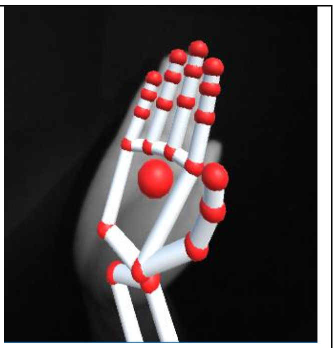
Yaw			

Table 1 - A test of the Leap Motion's tracking capabilities

Following this, I moved onto programming with the Leap Motion. As mentioned above, I decided to work in C++ due to the native support available for all of the devices I would be working with. However, the model used to access data from the Leap Motion is consistent throughout the SDKs so if there was a need to shift to another programming language, any work in C++ could be translated without much difficultly.

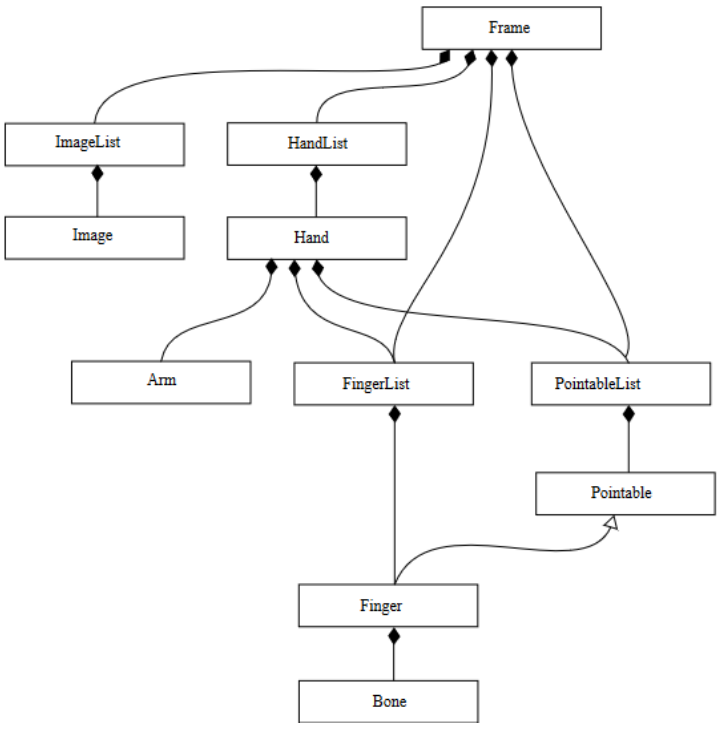


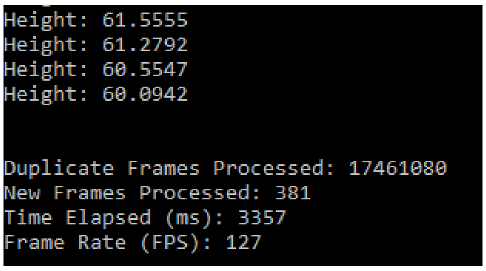
Figure 10 - Model used by the Leap Motion to access data (Colgan, 2014)

Like the Myo, the SDK for the Leap Motion was bundled with a sample application and links to ample documentation. The application demonstrated how to connect to the Leap Motion programmatically and access the data collected by the sensor. Without any tinkering, the application would simply dump very detailed and extensive tracking data into the console window, including tracking data for individual fingers.

While this showed that there is a large volume of data available to use, it was unnecessary for me to attempt to analyse all of it. Thus, I modified the scope of the program to output the following, basic readings:

- Palm rotation
    - Pitch

    - Roll
    - Yaw
- Palm height about the Leap Motion
- Hand grab strength



```
Height: 61.5555
Height: 61.2792
Height: 60.5547
Height: 60.0942


Duplicate Frames Processed: 17461080
New Frames Processed: 381
Time Elapsed (ms): 3357
Frame Rate (FPS): 127
```

Figure 11 - Example output from the Leap Motion application

I decided on these readings as I feel that they are readings that are very easy for the user to manipulate using the Leap Motion, whilst retaining a level of reliability in the readings. During my work on the application, I found that there were two methods of obtaining information from the Leap Motion device; listening or polling.

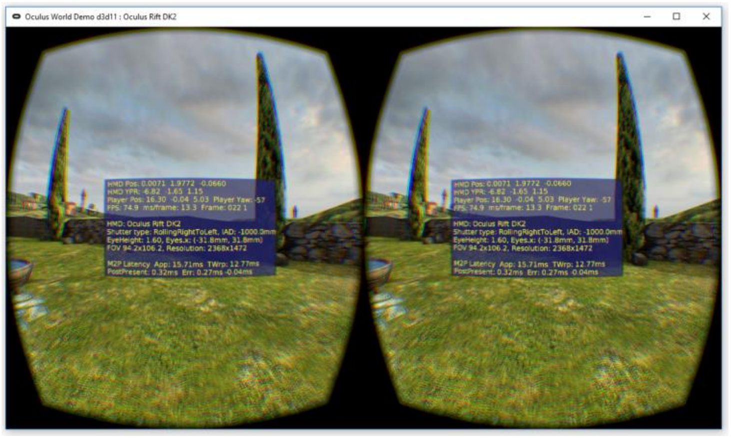
Using a listener meant an event was raised whenever a new frame was available from the device and you could write code to react to that event. This would allow me to process incoming frames very efficiently as I would be able to process the frame as and when it becomes available. However, this has the potential to introduce a threading issue into my program; if the output device is unable to cope with the current frame before a new frame is detected then there is a risk of creating multiple threads that are all blocked, waiting for access to the output device.

Using polling means that you would need to manually request a frame from the device. This frame is not guaranteed to be a new frame, thus you would need to implement some form of validation to check that you're not re-processing duplicated frames. I decided polling was a better option for me to use as I can fine-tune how the frames would be processed, despite the increased validation to check that I'm processing the correct frame.

3.1.3 Oculus Rift

I installed the software to run the Oculus Rift easily, however correctly wiring the device was much more difficult than the either the Myo or the Leap Motion. Multiple wires and connections were required between the PC and the Oculus.

Like the other devices, I experimented with some sample applications that were bundled with the Oculus SDK. One such application rendered a scene with a house and allowed me to move around the scene shown in Figure 12.



```
HMD Pos: 0.0071  1.9772  -0.0660
HMD YPR: -6.82   1.65  1.15
Player Pos: 16.30  -0.04  5.03  Player Yaw: -57
FPS: 74.9  ms/frame: 13.3  Frame: 022 1

HMD: Oculus Rift DK2
Shutter type: RollingRightToLeft, IAD: -1000.0mm
EyeHeight: 1.60, Eyes.x: (-31.8mm, 31.8mm)
FOV 94.2x106.2, Resolution: 2368x1472

M2P Latency  App: 15.71ms  TWrp: 12.77ms
PostPresent: 0.32ms  Err: 0.27ms -0.04ms
```

Figure 12 - Sample application for the Oculus Rift

The motion tracking functionality of the Oculus Rift was exceptional. I found that both the movement in space and rotations of my head about an axis were very accurately tracked, with the view into the scene being adjusted accordingly. Following this, I opened a debug menu in the application which allowed me to view the raw sensor readings which confirmed the accuracy of the tracking mechanism.

3.1.4 Conclusions

I found that the motion tracking capabilities of each device was largely exceptional. Both the Leap Motion and Oculus Rift could demonstrate that they could track motion to a high degree of accuracy. While the Myo could demonstrate that it could track rotational motion well, I was not convinced in the EMG-based gesture detection due to the errors in the gesture recognition within the software tutorial and the low-level analysis of the raw sensor values.

The stand-out device during this phase was the Leap Motion. I found that the device provided a middle ground of the three devices. It carries level of portability, only being connected to the PC with a single USB connection versus the Oculus with its various connections and the wireless functionality of the Myo. It also boasts a large volume of tracking data, which leaves potential to implement better control schemes later in the project. The documentation was very clear and there seems to be an active community of users who are currently working with the device.

3.2 Phase 2 - Output Devices

In the second phase of work, I wanted to assess the output devices available for the project, exploring how they could be made to collaborate with the other devices. As the feedback provided to the user through the system is an important aspect of a telepresence system, it was important to sample the potential feedback methods and make use of those that are appropriate.

3.2.1 Oculus Rift

While testing the Oculus in phase one, I also tested the output capabilities of the device; the ability to stream video. To do this, I sampled some video streaming applications available for free from the Oculus store. Whilst external factors such as internet speed, the quality of the application and personal effects (wearing glasses underneath the Oculus Rift) may have affected the quality of the video, I found that overall, the quality wasn't very good.

I found that using virtual projections (where the contents of the video would be rendered onto a screen in the virtual environment) to view a video would always result in a loss of video quality when compared to watching the video on a monitor. The only case where I found the experience of the virtual projection to be acceptable was when the quality of the original video was very high (1080p video). I also found that virtual projections were not very immersive, thus a better experience was provided through watching the video on a normal screen.

3.2.2 Nao Robot

I found that the Nao Robot is a very capable piece of kit but can also be very frustrating to work with. As I decided I would be working in C++, a Mac or Linux operating system was required to compile any applications for the robot (Aldebaran Robotics (b), 2015). This meant developing with the robot would be slower than developing with the other devices as I had little prior experience working with these operating systems.

Before trying to program for the robot, I wanted to explore the Choregraphe application that can interface with the robot. However, I found that there were network issues that prevented the robot connecting to the application wirelessly. It was unclear what those issues were but it seemed to be caused by a faulty router. It was possible to connect to the robot wirelessly by manually restarting the router by turning it off and on but the wireless connection was not very stable. Thus, connecting to the robot with an ethernet cable was required to be able to work with it in Choregraphe.

With Choregraphe, I sampled how the robot could move. I could use the drag and drop interface to make the robot perform simple movement commands. However, I found that the robot sometimes suffered from balance issues, occasionally falling over when walking or turning too quickly. Also, I sampled the audio commands available in Choregraphe, making the robot speak after performing specific commands. Using Choregraphe in this experimental manner was helpful to get a broad idea of the low-level functionality of the robot.

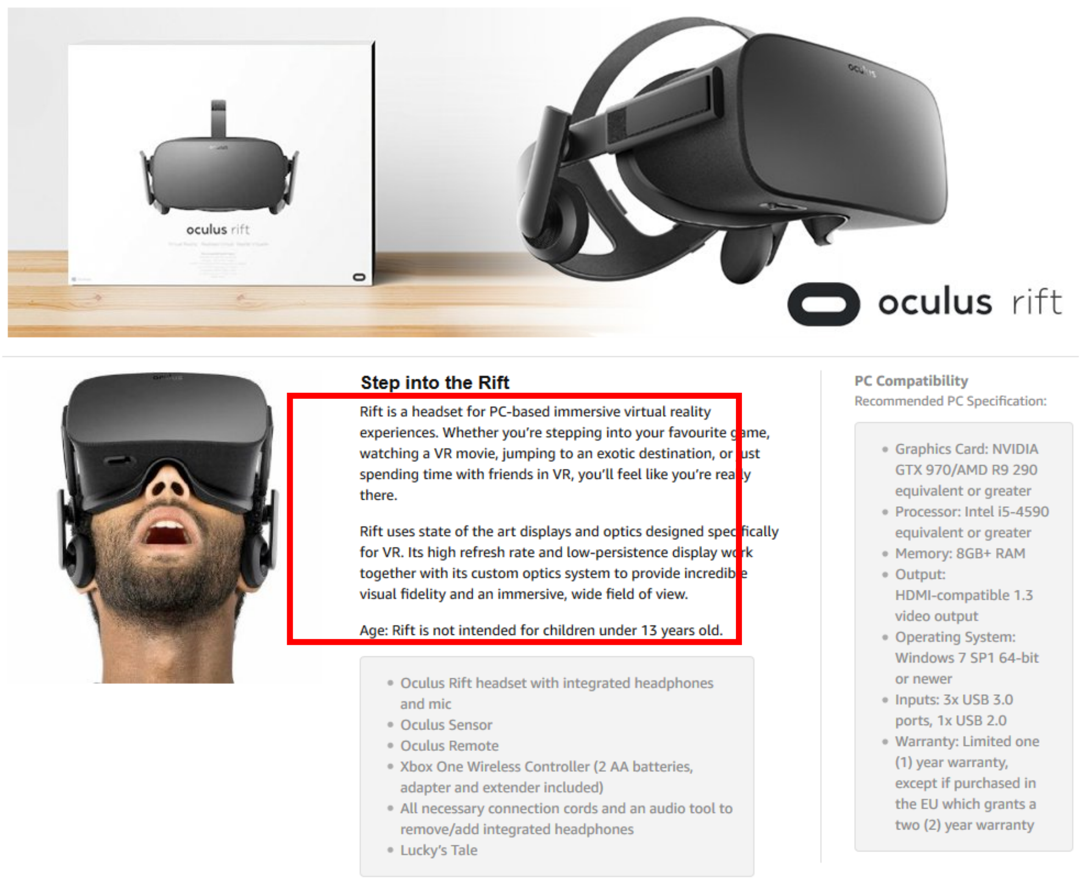
Despite this, I created a virtual box running Ubuntu, and compiled a sample application from the C++ SDK for the robot. I found myself faced with many challenges before being able to compile the sample application. To begin with, the application had to be compiled using a very specific method. Despite the method being documented, I found that I would hit errors due to differences in the development environment. After some weeks of debugging, the program compiled correctly. However, the next challenge was to connect the robot to application. Because I was using a virtual box, this proved to be more difficult than I anticipated. I was able to diagnose the type of network connection the robot was using with the PC and was able to pass the connection onto the virtual box.

At this point, I could begin to modify the sample application to begin exploring how I could interface with the robot. I found that the simplest way to do this was to use a specialised proxy which allowed access to pre-programmed functions within the SDK. I found that there existed proxies that I could utilise to allow me to access functions to move to robot, thus I modified the sample application to move the head of the robot based on some keyboard input.

This seemed to work well but was very primitive in its functionality. If the user pressed ‘w’ the robots head would rotate to face upwards, if the user pressed ‘s’, the inverse would happen.

3.2.3 Conclusions

I found that the Oculus Rift has potential to be a very capable output device. This is supported by the fact that consumer VR devices are largely marketed by touting the output specifications of the device, which could be taken to indicate that the Oculus is largely an output-focussed device.



Step into the Rift

Rift is a headset for PC-based immersive virtual reality experiences. Whether you're stepping into your favourite game, watching a VR movie, jumping to an exotic destination, or just spending time with friends in VR, you'll feel like you're really there.

Rift uses state of the art displays and optics designed specifically for VR. Its high refresh rate and low-persistence display work together with its custom optics system to provide incredible visual fidelity and an immersive, wide field of view.

Age: Rift is not intended for children under 13 years old.

- Oculus Rift headset with integrated headphones and mic
- Oculus Sensor
- Oculus Remote
- Xbox One Wireless Controller (2 AA batteries, adapter and extender included)
- All necessary connection cords and an audio tool to remove/add integrated headphones
- Lucky's Tale

PC Compatibility
Recommended PC Specification:

- Graphics Card: NVIDIA GTX 970/AMD R9 290 equivalent or greater
- Processor: Intel i5-4590 equivalent or greater
- Memory: 8GB+ RAM
- Output: HDMI-compatible 1.3 video output
- Operating System: Windows 7 SP1 64-bit or newer
- Inputs: 3x USB 3.0 ports, 1x USB 2.0
- Warranty: Limited one (1) year warranty, except if purchased in the EU which grants a two (2) year warranty

Figure 13 - Marketing of the Oculus Rift (Amazon.co.uk, 2017)

Whilst the tracking technology embedded in the device is important, it appears the primary focus of the device has been its output capabilities. However, for the Oculus to create a good viewing experience, I found that the video source should be 1080p or better. Anything less meant that the experience was bad enough to potentially cause health issues.

I found that the decision to work in C++ made working with the Nao robot much more difficult. The decision meant that the development must be completed in an unfamiliar environment and the work I had completed with the input devices would need to be ported over and compiled in the Ubuntu environment I was working with to be able to develop the final project deliverable.

However, I also learnt that the architecture used in the SDK was simple to use; creating a proxy and sending commands to the robot through that proxy was a simple process. Due to this, upon the completion of this phase, I could envision mapping the data from the input devices to the robot's movement.

3.3 Phase 3 - Project Deliverable

3.3.1 Final Design Decisions

Following the conclusion of the first two phases, multiple system design decisions had been made.

- The system will be built with the collaboration of Nao robot and the Leap Motion.
    - I made the decision to exclude the Myo armband as the data provided by the Myo armband was eclipsed by the data provided by the Leap Motion. I decided that the wireless access to rotational data alone was not enough to justify its use when I could instead use the Leap Motion and be provided with a much wider array of data.

    - I made the decision to exclude the Oculus Rift as the cameras on the Nao robot cannot provide a high enough quality video stream to provide a meaningful virtual experience. Without this, the Oculus would be limited to be used as an input device, which is unjustified when the Leap Motion can provide a similar tracking experience, whilst eliminating the need to tether the user to the PC with the multiple wires required for the Oculus Rift.
- The system will allow the user to use their hand to control the Nao robot.
    - The user will be able to freely control the robot while it walks in an open space. The user will also be able to control the speed at which the robot walks.
    - The user will be able control the head of the robot, matching the orientation of the robot's head to the user's hand.
    - The user will be able send a string to text to the robot and the robot will say the words.
    - The user will be able to control the arms of the robot
- The system will be built in C++ in an Ubuntu environment.
- The system will implement a user interface which allows the user to select which body part the user wants to control.
- The system will record the commands sent to the robot and will allow the user to play back those commands.
    - The system will allow the user to save and load commands to file.
    - The system will allow the user to edit the order in which a command would be executed.

3.3.2 Network Diagram

For the system to be able to control the robot, the devices must be connected in some configuration. Figure 14 describes the configuration used.

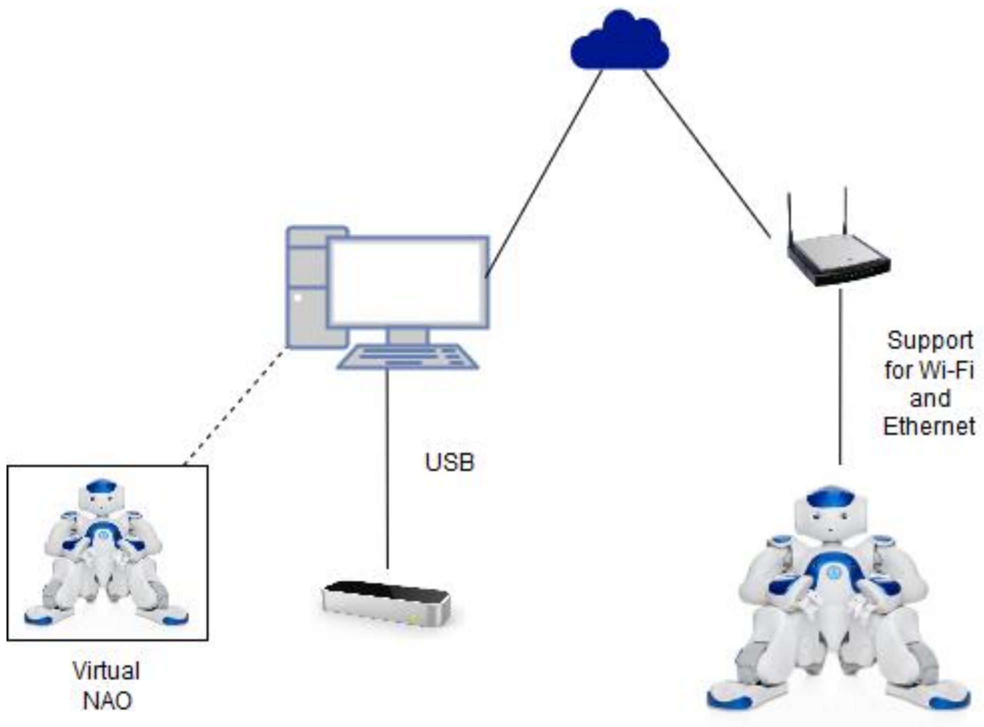


Figure 14 - Network Diagram

This configuration allows for the robot to be untethered while being controlled by the system. However, this depends on a successful and stable Wi-Fi connection, which I have found to be unreliable. During development, I either utilised an ethernet cable to ensure that the connection to the robot was stable, or used a virtual robot created using the Choregraphe suite.

3.3.3 System Architecture

The system is built using the architecture described in Figure 15.

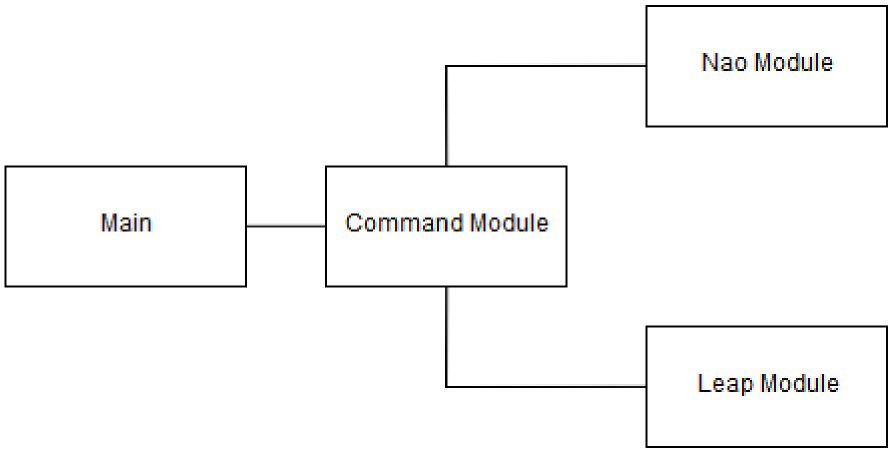


Figure 15 - System Architecture

The main module handles the user interface creation, events, and destruction. It has links to functions in the command module which handles the core of the system, the communication between the devices. The remaining modules interface directly with the devices and have been separated to ensure that there is no unwanted crossover between the devices. Also, this architecture is easier to maintain and understand than in a flat architecture, where there is no clear segmentation of functionality.

3.3.4 Implementation

In implementing this system, the first challenge I was faced with was porting the work I had completed with the Leap Motion in phase one into an Ubuntu development environment. This proved to be very challenging, with the Leap Motion support for the Ubuntu environment lacking. I installed a version of the Leap Motion software required to connect with the sensor onto the virtual box I was working on and tested the device using the visualizer, however I noticed there was a drop in the frames being processed per second. This was caused by the lower processing power allocated to the virtual box and the decreased bandwidth of the USB connection being passed onto the virtual box. However, in this state, the sensor would still function as a proof of concept for the system in the Ubuntu environment, so I left this issue as one to solve later.

After installing the Leap Motion software and verifying the connection, I attempted to integrate the Leap Motion SDK with the sample application for the Nao robot. This brought up errors in the project compilation and prevented the development of the system. The Leap Motion SDK would not be recognised by the compiler, despite my efforts to add references to the SDK in the compiler.

After spending a large amount of time attempting to troubleshoot this, I realised that my decision to work in C++ had been influenced by the fact that I would be using multiple devices and C++ was the only language with native support on all the devices I would be using. However, due to deciding to exclude the Oculus Rift from the project, I found that Python was also a common language between the two devices I would be using.

Thus, I decided to shift the system to be written in Python, having to rewrite the code completed during phase one and two in Python. Fortunately, this was not too difficult as the SDKs in Python have a similar structure to the ones in C++, so all of what I learnt during the first two phases was still transferred into the system. Also, it meant that the development could continue in the Windows OS which I am much more comfortable with and have more experience working in. As a side effect of this, the issues I saw with the reduced number of frames in the virtual box were also solved.

I compiled separate projects which interfaced with the Leap Motion and Nao robot separately. I found that the complicated compilation process required for the Nao robot was much simpler in Python, with the method to import the Leap Motion SDK also being much simpler. With these two smaller projects complete, I then went onto testing compiling a project with had access to both the Leap Motion SDK and the Nao SDK, which I was also able to compile very easily.

With the compilation issues out of the way, I was then able to focus on building the core of the application. I started by building a GUI using a UI creation tool for python called QT Designer. This tool along with the PyQT Python plugin allowed me to quickly build and implement a GUI in my application.

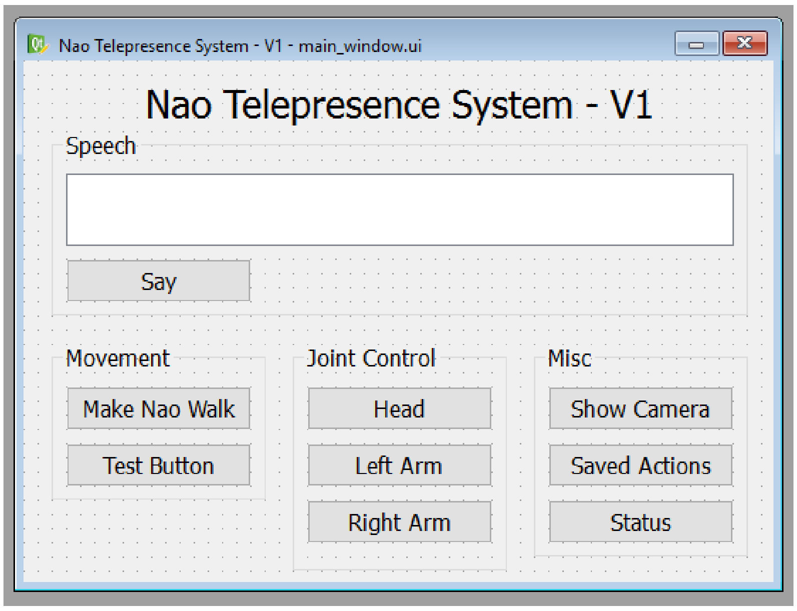


Figure 16 - A version of the main window

Once parts of the GUI had been completed, I started to add functionality to the system. I developed and implemented an algorithm that would handle the input from the Leap Motion and pass it onto the Nao robot.

```
ALGORITHM
An algorithm to take the raw input from the LM and pass it onto the Nao

      while no hands detected:
            Check whether hand is detected

      while hand extended
            Get hand height from LM
            Scale hand height to LM input range (30-90)

            Get hand rotation angles from LM

            Process hand rotation angles
                  - Scale according to hand height using -scaled height
and height as limiters
                  - Any further processing

            Scale hand rotation angles to output limits (0 - 99)
            Output hand rotation angles to GUI

            Pass hand rotation angles to Nao

            Append hand rotation angles to the local list of actions
            Check whether hand is extended

      If actions have been added to the local action list
            Record local action list in a global action list
```

Figure 17 - The algorithm used to implement control

Using this algorithm meant that the method used to implement the functionality across different parts of the robot was consistent. This consistency meant that the input used was consistent and could be saved, allowing for the action to be replicated without the need to replicate the input.

Mentioned in Figure 17 is the ability to scale the input to a specified range. This was required to be able to map the raw input from the Leap Motion to the output, the movement of the Nao robot. Figure 18 shows the method used to implement the scale function.

```python
def scale(value, old_min, old_max, new_min, new_max):
    old_range = old_max - old_min
    new_range = new_max - new_min
    new_value = (((value - old_min) * new_range) / old_range) + new_min
    if new_value > new_max:
        new_value = new_max
    if new_value < new_min:
        new_value = new_min
    return new_value
```

Figure 18 - Code snippet showing the scale function

Implemented in the scale function are two if statements to ensure that the scaled value falls within the range specified, forcing the value to the max or min if it doesn't. This was required due to some cases where the input value was outside the max and min range specified, for example, if the hand height value was higher than the maximum hand height value specified in the system.

This scale function has mainly been used to scale the input parameters according to how high the users hand is above the Leap Motion. This allows them to control how sensitivity of the system, i.e. how much a change in their hand orientation will affect the robot.

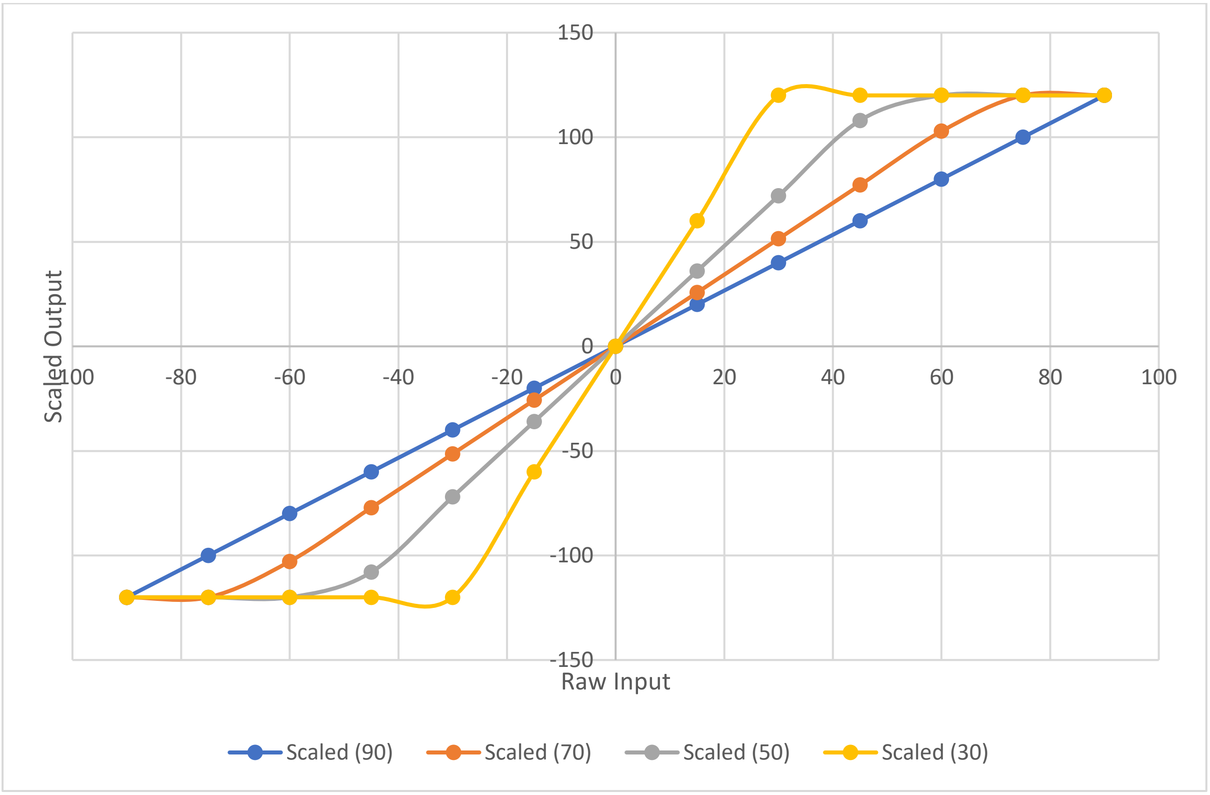


Figure 19 - Graph showing how the hand height affects how the output is scaled

This is implemented by scaling the raw input parameters twice, once using the scaled height value as limiters (positive and negative for maximum and minimum respectively), and again using the movement range limits as limiters, as defined in the Nao documentation (Aldebaran Robotics (c), 2015). In Figure 19, the gradient of the line showing the scaled output is connected to the sensitivity; the higher the gradient, the lower the sensitivity and the higher the impact the user's motion will have on the robot.

The method to process the input depends on the body part being controlled by that function. In this section I have explained how the input has been processed for each function.

Head

The input consists of the pitch and yaw value describing the orientation of the head in the two axes respectively. Each value is scaled according to the height to give the user the ability to control how much their motion will affect the robot's movement. The values are then scaled according to the motion limitations as defined in the Nao documentation (Aldebaran Robotics (c), 2015). For the head, there exists a shifting motion range available for the pitch of the head, depending on the yaw of the head. This is due to possible collisions occurring between the robot's head and shoulders.

To account for this, the range the pitch value is scaled to is calculated depending on the yaw value. There is no set relationship between the yaw value and range of pitch movement available as the relationship is based on the physical space available around the robot. However, I modelled the relationship using excels graphing tools to create a polynomial formula which works as an estimation of the relationship.

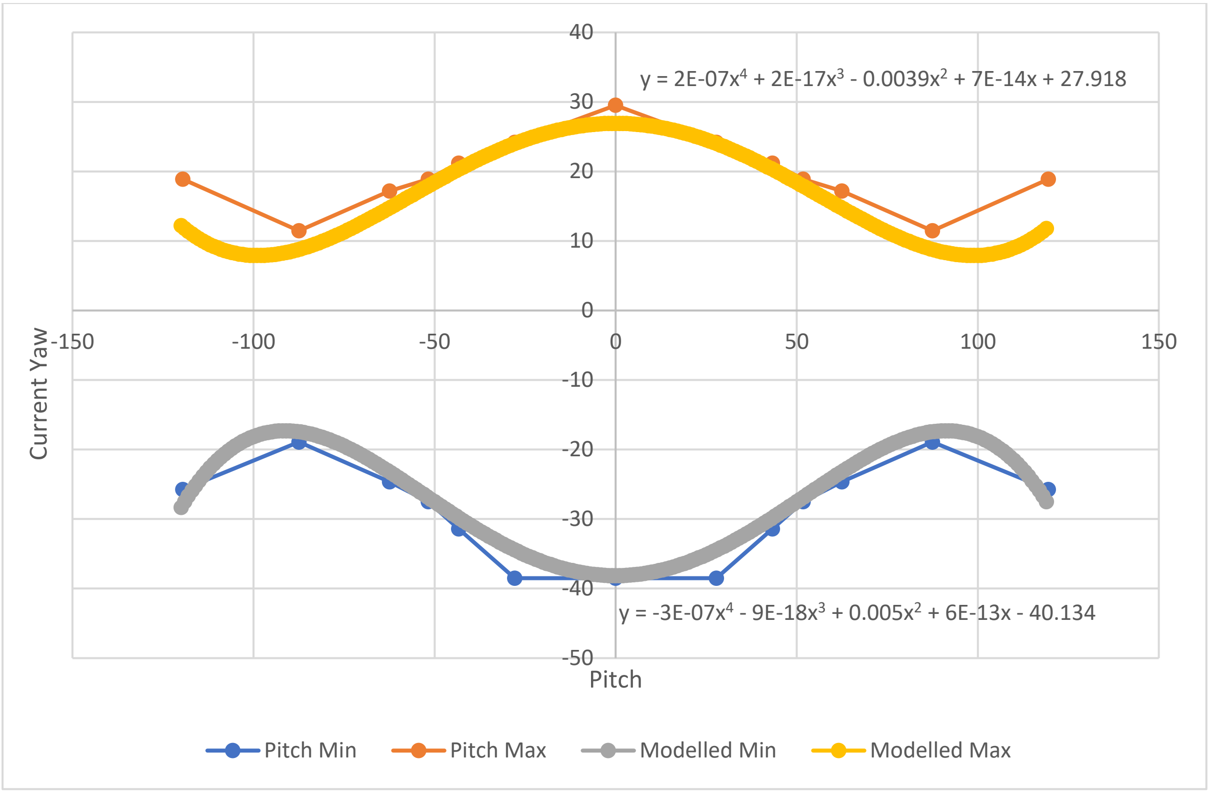


Figure 20 - Graph showing the polynomial formula and the fit of the model to the actual range

The model was created by plotting the range on a graph then using excels graphing tools to draw a curve of best fit through the points. The formula of that line was then used as a model. To check that there weren't any points at which the modelled range would fall outside of the absolute range as specified in the documentation, I plotted the range produced by the model. There are points at the ends of the curve that fall outside of the absolute range specified in the documentation, but these values are filtered out during the first scaling process.

Arms

In the Nao's arms, there are three joints at which the arm can be controlled; the shoulder, elbow and wrist. The mapping of input to output is shown in Table 2.

Table 2 - Table showing the mapping from input axis to output axis

Hand Gesture	Joint	Input Axis	Output Axis	Shown on Axis
Hand fully extended	Shoulder	Pitch	Pitch	Vertical
		Roll	Roll	Horizontal
Middle three fingers extended	Elbow	Roll	Roll	Vertical
		Yaw	Yaw	Horizontal
Outer two fingers extended	Wrist	Roll	Roll	Horizontal

The input from the Leap Motion is scaled to different limits depending on which joint the user wants to control. They can select the joint using a different gesture with their hand, with the robot arm moving to match the movement of the user's hand. The two different arms can each be individually controlled using the buttons on the main window.

Movement
Movement, defined in this scope as the robot's ability to walk around in space, functions slightly differently to the other motion functions implemented in the system. Instead of the robot matching the motion of the user's hand, the user can control the robot's movements by using their hand's orientation as a slider.

By increasing the pitch of the user's hand, they can make the robot walk forwards, the greater the angle they make with their hand in the clockwise direction, the faster the robot will walk. Inversely, if they rotate their hand in the anticlockwise direction, the robot will begin to walk backwards, walking quicker the bigger the angle they make.

The same is also true with the yaw angle of the user's hand; the user can make the robot turn clockwise, or anticlockwise, depending on how they orient their hand in the yaw direction.

Also, the user can stop the robot walking by keeping their hand in a neutral position, where both the yaw and pitch would be zero. To assist the user in tracking the orientation of their hand, sliders have been implemented in the GUI as shown in Figure 21. These sliders will update in real-time to show the user the orientation of their hand.

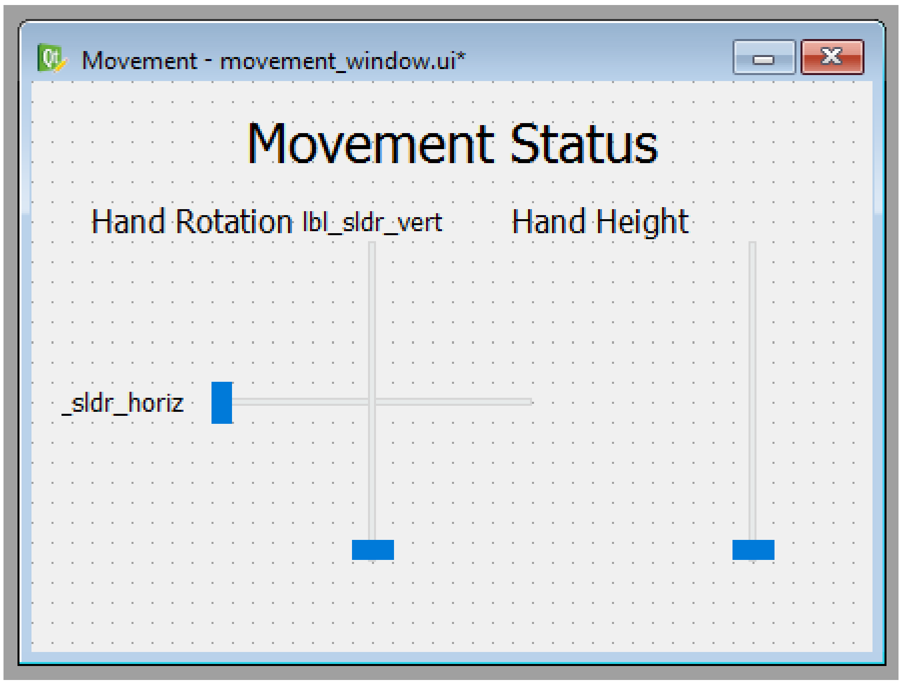


Figure 21 - A version of the movement window

Speech
The user has the ability to make the Nao speak. This has been implemented by adding a box on the main window where the user can enter text. The Nao has a built-in text-to-speech engine which has been utilised to make the Nao parse the text.

Vision
The user has the ability to use the Nao's cameras to create a live video feed. Due to connection limitations with the Nao, the user also can select the quality of the video feed. The quality ranges from an 80 x 60 video feed to a 320 x 240 video feed. My testing indicated that a higher quality would result in a lower quality video feed as the network bandwidth struggles to produce the frame rate required to maintain the video quality.

This function has been implemented by utilising the Nao video proxy. This works by sending the proxy the desired resolution, colour settings and max frames per second with the proxy returning a subscription to the video feed. An event is then raised whenever a new frame is available and handling the event to process the frame and output it on screen creates a video feed.

Status

The user can view the status of connected devices in the status window. This window can be used to troubleshoot issues with the devices and will display whether each device is connected and functioning properly. From this window, the user is also able to change the volume of the Nao robot, Check the Nao's battery level and see the bandwidth status in frames processed per second of the Leap Motion.

Saved Actions

The system will record any actions they send to the Nao robot while the system is running and the user can play back these actions. These actions are saved in a block of a single joint motion and are recorded whenever the user completes the motion. The user can save the actions to a file which is required for the actions to persist beyond the lifetime of the system running. The user can then reload these files, where the system will parse the file and populate the list of actions saved in the system. The system uses a standard file type, JSON, to save actions. While a method to modify a block of actions is not implemented, due to the use of a JSON file type, there exists many third-party applications to modify these files.

The information saved about an individual action is the joint it affects and the parameters sent to the Nao to carry out the action. After each frame of input data, the action is recorded in a local actions list. Once the user has completed the motion, the local list is recorded in a master list. A summary of the master list can be viewed through the actions list window.

```
def print_action_list(window):
    list_view = window.wgt_command_list
    list_view.clear()
    fps = leapModule.get_bandwidth_status()
    for local_action_list in action_list:
        if local_action_list[0]["type"] in ("left_shoulder", "left_elbow",
"left_wrist"):
            output_string = "left arm - " +
"{:.2f}".format(len(local_action_list)/fps) + "s"
        elif local_action_list[0]["type"] in ("right_shoulder",
"right_elbow", "right_wrist"):
            output_string = "right arm - " +
"{:.2f}".format(len(local_action_list)/fps) + "s"
        else:
            output_string = local_action_list[0]["type"] + " - " +
"{:.2f}".format(len(local_action_list)/fps) + "s"
        list_view.addItem(output_string)
```

Figure 22 - Code snippet showing how the action list can be accessed

Chapter 4 - Evaluation

To evaluate the performance of the system, I will refer to the objectives written earlier in the report:

1. **The user will be able to control the movement of the robot in 3D space, using some gesture as the input.**

This objective has been met as the system allows the user to make the robot walk around a room, using a hand gesture as the input to the system.

2. **The user will be able to control the movement of the head of the robot, using some gesture as the input.**

This objective has been met as the system allows the user to control the robots head, using a hand gesture as the input to the system.

3. **The user will be able to control each arm of the robot individually. The arm should be able to be controlled to a level where the robot is able to interact with the environment.**

This objective has been partially met. The system allows the user to control the arms of the robot using a hand gesture as input to the system, but the level of control the user has over the arms of the robot may not be high enough to be able to usefully interact with an environment. This is caused by the fact that the system only allows a single joint in a single arm to be controlled at any time.

4. **The user will be able to utilise the speakers on the robot to relay some message.**

This objective has been met as the system allows the user to enter a phrase and send the phrase to the robot for it to speak.

5. **The user should be able to get feedback from the robot, in the form of a camera image.**

This objective has been met as the system allows the user to access a video feed from the camera on the robot. The quality of the video feed is dependent on the quality of the network connection; however, this is a factor external to the system.

6. **The user should be able to control the robot while it's untethered.**

This objective has been partially met. The system supports a wireless connection to the robot but the connection is not stable. However, as the factor that causes this is external to the system, this objective could be complete.

7. **The motions required to move the robot will be relatable to the motion made by the robot.**

This objective has been met as the all motions required to move the robot are connected in some form to the motion that the robot will make.

8. **The movement of the robot should be controlled and measured; any erratic movements or latency should be minimised.**

This objective has been partially met. The system has measures in place to minimise erratic movements, specifically, when the robot is walking the maximum speed is kept low. However, despite this, erratic motions and latency can be caused by connection issues in the robot.

9. **The delivered project should be usable by any person with minimal training.**

This objective has been met. There are no complicated motions required to operate the system. Some background knowledge will be required to be able to connect with the robot.

10. **The delivered project should run on either Windows, Mac, or Ubuntu.**

This objective has been met, the system has been fully tested on a Windows PC.

11. **The delivered project may require a PC capable of utilising the Oculus Rift hardware.**

As the Oculus Rift was not used in the final deliverable, this objective is no longer valid.

4.1 Testing

The testing carried out on the system has been documented below.

#	Description	Expected Result	Actual Result	Comment
1	Running the application	When the application is run, the main window should open.	When the application is run, the main window opens.	Passed
2	Button event, Say	When the "Say" button is clicked with some text entered in the text box, a connected robot should speak the text.	When the "say" button is clicked with some text entered in the text box, a connected robot speaks the text.	Passed
3	Button event, Walk	When the "Walk" button is clicked, the movement window should open.	When the "Walk" button is clicked, the movement window opens.	Passed
4	Button event, Head	When the "Head" button is clicked, the movement window should open.	When the "Head" button is clicked, the movement window opens.	Passed
5	Button event, Left arm	When the "Left Arm" button is clicked, the movement window should open.	When the "Left Arm" button is clicked, the movement window opens.	Passed
6	Button event, Right arm	When the "Right Arm" button is clicked, the movement window should open.	When the "Right Arm" button is clicked, the movement window opens.	Passed
7	Button event, Show camera	When the "Show Camera" button is clicked, the camera window should open.	When the "Show Camera" button is clicked, the camera window opens.	Passed
8	Button event, Saved actions	When the "Saved Actions" button is clicked, the saved actions window should open.	When the "Saved Actions" button is clicked, the saved actions window opens.	Passed
9	Button event, Status	When the "Status" button is clicked, the status window should open.	When the "Status" button is clicked, the status window opens.	Passed
10	Functionality, Walk	When the movement window is opened, some text should be shown on the window indicating that the application is waiting	When the movement window is opened, some text is shown on the window indicating that the application is waiting for	Passed

		for some input.	some input.	
11	Functionality, Walk	When the user places their hand above the Leap Motion, the text should be replaced with three sliders which update in real-time.	When the user places their hand above the Leap Motion, the text is replaced with three sliders which update in real-time.	Passed
12	Functionality, Walk	As the user changes the orientation of their hand in the yaw and pitch direction, the robot should move accordingly.	As the user changes the orientation of their hand in the yaw and pitch direction, the robot moves accordingly.	Passed
13	Functionality, Walk	As the user raises their hand the amount their motions affect the robot should be scaled to the height of their hand.	As the user raises their hand the amount their motions affect the robot is scaled to the height of their hand.	Passed
14	Functionality, Head	When the movement window is opened, some text should be shown on the window indicating that the application is waiting for some input.	When the movement window is opened, some text is shown on the window indicating that the application is waiting for some input.	Passed
15	Functionality, Head	When the user places their hand above the Leap Motion, the text should be replaced with three sliders which update in real-time.	When the user places their hand above the Leap Motion, the text is replaced with three sliders which update in real-time.	Passed
16	Functionality, Head	As the user changes the orientation of their hand in the yaw and pitch direction, the robot should move its head to match the user’s hand accordingly.	As the user changes the orientation of their hand in the yaw and pitch direction, the robot should move its head to match the user’s hand accordingly.	Passed
17	Functionality, Head	As the user raises their hand the amount their motions affect the robot should be scaled to the height of their hand.	As the user raises their hand the amount their motions affect the robot is scaled to the height of their hand.	Passed
18	Functionality, Left/Right Arm	When the movement window is opened, some text should be shown on the window indicating that the application is waiting for some input.	When the movement window is opened, some text is shown on the window indicating that the application is waiting for some input.	Passed
19	Functionality, Left/Right Arm	When the user places their hand above the Leap Motion, the text should be replaced with three sliders which update in real-time.	When the user places their hand above the Leap Motion, the text is replaced with three sliders which update in real-time.	Passed
20	Functionality,	As the user changes the	As the user changes the	Passed

	Left/Right Arm	orientation of their hand, the robot should move its arm accordingly.	orientation of their hand, the robot moves its arm accordingly.	
21	Functionality, Left/Right Arm	As the user raises their hand the amount their motions affect the robot should be scaled to the height of their hand.	As the user raises their hand the amount their motions affect the robot is scaled to the height of their hand.	Passed
22	Functionality, Left/Right Arm	The arm which is controlled by the user should correspond with the arm button the user has clicked on the main window.	The arm which is controlled by the user corresponds with the arm button the user has clicked on the main window.	Passed
23	Functionality, Left/Right Arm	When the users hand gesture is an extended hand gesture, the user should be able to control the shoulder joint.	When the users hand gesture is an extended hand gesture, the user can to control the shoulder joint.	Passed
24	Functionality, Left/Right Arm	When the users hand gesture is a middle three fingers extended hand gesture, the user should be able to control the elbow joint.	When the users hand gesture is a middle three fingers extended hand gesture, the user can to control the elbow joint.	Passed
25	Functionality, Left/Right Arm	When the users hand gesture is an outer two fingers extended hand gesture, the user should be able to control the wrist joint.	When the users hand gesture is an outer two fingers extended hand gesture, the user can to control the wrist joint.	Passed
26	Functionality, Camera	When connected to a real robot, a window should be opened showing a video feed from the Nao's camera.	When connected to a real robot, a window is opened showing a video feed from the Nao's camera.	Passed
27	Functionality, Camera	The quality of the video feed should match the quality selected in the status window.	The quality of the video feed matches the quality selected in the status window.	Passed
28	Functionality, Saved Actions	When the saved actions window is opened, a summary of any actions that have been carried out during the lifetime of the application should be listed.	When the saved actions window is opened, a summary of any actions that have been carried out during the lifetime of the application is be listed.	Passed
29	Functionality. Saved Actions	When the user selects an action and clicks Play Selected Action, the action should be replayed on a connected robot.	When the user selects an action and clicks Play Selected Action, the action is replayed on a connected robot.	Passed

30	Functionality, Saved Actions	When the user clicks on Play All Actions, all recorded actions should be replayed on a connected robot in the order they are listed.	When the user clicks on Play All Actions, all recorded actions are replayed on a connected robot in the order they are listed.	Passed
31	Functionality, Saved Actions	When the user selects an action and clicks Move Action Up or Down, the selected block of actions should be moved up or down the list by one place.	When the user selects an action and clicks Move Action Up or Down, the selected block of actions is moved up or down the list by one place.	Passed
32	Functionality, Saved Actions	When test x is repeated after an action block has been moved, test 30 should still evaluate correctly.	When test x is repeated after an action block has been moved, test 30 still evaluates correctly.	Passed
33	Functionality, Saved Actions	When the user selects an action and clicks Delete Action, the action should be deleted from the list of actions shown.	When the user selects an action and clicks Delete Action, the action is deleted from the list of actions shown.	Passed
34	Functionality, Saved Actions	When the user clicks Save Actions, a dialog should be opened allowing the user to save the actions to a file.	When the user clicks Save Actions, a dialog is opened allowing the user to save the actions to a file.	Passed
35	Functionality, Saved Actions	When the user clicks Load Actions, a dialog should be opened allowing the user to open a previously saved actions file and the actions from that file should be loaded into the list.	When the user clicks Load Actions, a dialog is opened allowing the user to open a previously saved actions file and the actions from that file should be loaded into the list.	Passed
36	Functionality, Saved Actions	When the user tried to load an invalid file, the system should handle the error gracefully.	The system crashed.	Failed. Validation was put in place to handle the error. Upon retesting the system passed the test.
37	Functionality, Status	When the status window is open, the status of the Leap Motion sensor should be shown.	When the status window is open, the status of the Leap Motion sensor is shown.	Passed
38	Functionality, Status	When the status window is open, the bandwidth (frame rate) of the Leap Motion should be continually updated	When the status window is open, the bandwidth (frame rate) of the Leap Motion is continually updated	Passed
39	Functionality, Status	When the status window is open, the status of the Nao robot should be shown.	When the status window is open, the status of the Nao robot is shown.	Passed

40	Functionality, Status	If a Nao robot is connected, the user should be able to set the volume of the robot using the slider.	If a Nao robot is connected, the user can set the volume of the robot using the slider.	Passed
41	Functionality, Status	The user should be able to set the IP address and port used to connect to the Nao robot by entering the respective values and clicking save.	The user can set the IP address and port used to connect to the Nao robot by entering the respective values and clicking save.	Passed

Chapter 5 - Conclusion

5.1 Recommendations

My recommendations for an attempt at this project would be to spend more time early in the project exploring the different options available to interface between the devices. One of the main challenges in the project is the collaboration of different devices and by leaving this aspect for a late stage in the project, I was limited in the amount of functionality I could implement in the system. My recommendation would be to take a proof of concept approach and implement small, barely working applications to prove a method is available.

I would also recommend a deeper introduction to the project. Although an aspect of self-learning is required for the project, I would have benefitted from an introduction to the deliverable from the previous attempt at the project. This would have set me up better to improve upon the previous version of the project, giving me concrete aims to target with my target deliverable.

5.2 Future Work

For the continuation of the project, I would recommend the following improvements that could be made:

- Improving the control scheme for the arms of the robot.
    - Currently the arms of the robot are relatively difficult to control. Ideally the user should be able to match the motion of the robot to their own motion easily, however, accurately controlling the elbows of the robot using wrist motions can be difficult.
    - One possible solution could be to utilise the rotation in the users elbow to control the elbow on the robot. This should be possible with the Leap Motion as it supports elbow visualisations, indicating that data on the motion of the elbow is available through the SDK.
- Improving the network connection to the robot.
    - While technically external to the project deliverable, it has a direct impact on the performance on the deliverable. The inability to control secure a reliable wireless connection to the robot is detrimental to the project. An investigation into the hardware causing the network issues may result in a better performing system.
- Improving the user’s ability to modify actions they have saved in the past.
    - Currently, the user can modify a list of action blocks, moving each block up and down in the list to change the order in which the list of blocks would be executed. This could be improved by allowing the user to edit each block of action, allowing them to fine tune the actions they have recorded to a lower level.
    - While this is technically possible using a third-party JSON editor, it would be nice to keep the functionality in-house and would also prevent any potential data structure issues the user may face.
- Improving feedback provided by the robot
    - Currently the only form of feedback provided to the user is through the video stream provided by the camera. The Nao robot has the functionality to provide a much richer stream of feedback to the user than has been implemented. In improving the feedback provided to the user, the system would be brought closer to a functional telepresence system.

References

Aldebaran - Softbank Robotics. (2014, May). *About us - Gallery*. Retrieved March 2017, from Softbank Robotics: https://www.ald.softbankrobotics.com/en/press/gallery/nao

Aldebaran Documentation (a). (2016). *Nao H25*. Retrieved April 2017, from Aldebaran Documentation: http://doc.aldebaran.com/2-4/family/nao_h25/index_h25.html

Aldebaran Documentation (b). (2016). *Video Camera*. Retrieved April 2017, from Aldebaran Documentation: http://doc.aldebaran.com/2-4/family/robots/video_robot.html

Aldebaran Robotics (a). (2015, 10 06). *Aldebaran - Nao Developer Docs (a)*. Retrieved January 2017, from Aldebaran: http://doc.aldebaran.com/2-1/news/index.html

Aldebaran Robotics (b). (2015). *Aldebaran - Nao Developer Docs (b)*. Retrieved January 2017, from Aldebaran: http://doc.aldebaran.com/2-1/dev/cpp/index.html

Aldebaran Robotics (c). (2015, 10 06). *Aldebaran - Nao Developer Docs (c)*. Retrieved April 2017, from Aldebaran: http://doc.aldebaran.com/2-4/family/robots/joints_robot.html

Amazon.co.uk. (2017). *Amazon - Oculus Rift*. Retrieved April 2017, from Amazon.co.uk.

Bernhardt, P. (2015, March 14). *Myo - Blogs*. Retrieved January 2017, from Myo: http://developerblog.myo.com/myocraft-logging-imu-and-raw-emg-data/

Cambridge Dictionary (a). (2013). *Meaning of "presence" in the English Dictionary*. Retrieved April 2017, from Cambridge.org: http://dictionary.cambridge.org/dictionary/english/presence

Cambridge Dictionary (b). (2013). *Meaning of "tele-" in the English Dictionary*. Retrieved April 2017, from cambridge.org: http://dictionary.cambridge.org/dictionary/english/tele

Colgan, A. (2014, August 9). *Leap Motion - Blogs*. Retrieved January 2017, from Leap Motion: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

Dean Beeler, A. G. (2016, March 25). *Oculus - Developer Blogs*. Retrieved January 2017, from Oculus: https://developer3.oculus.com/blog/asynchronous-timewarp-on-oculus-rift/

Deutsch, J. D., Borbely, M., Filler, J., & Karen, H. (2008). Use of a Low-Cost, Commercially. *Physical Therapy, 88*, 1-10. Retrieved April 2017, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.970.1918&rep=rep1&type=pdf

Hutchinson, L. (2014, March). *Oculus Rift Dev Kit 2 launches with 960×1080 resolution, lower latency*. Retrieved April 2017, from arstechnica: https://cdn.arstechnica.net/wp-content/uploads/2014/03/9464.jpg

Leap Motion Inc (a). (2017, January). *Leap Motion - Developer Docs (a)*. Retrieved January 2017, from Leap Motion: https://developer.leapmotion.com/documentation/javascript/api/Leap.Controller.html

Leap Motion Inc. (2015). *Leap Motion*. Retrieved April 2017, from Leap Motion Store.

Leap Motion Inc. (2017, January). *Leap Motion - Developer Docs (a)*. Retrieved January 2017, from Leap Motion: https://developer.leapmotion.com/documentation/javascript/api/Leap.Controller.html

Leap Motion Inc. (2017, January). *Leap Motion - Developer Docs (b)*. Retrieved January 2017, from Leap Motion: https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Hand.html

Leap Motion Inc. (2017, January). *Leap Motion - Developer Docs (c)*. Retrieved January 2017, from Leap Motion: https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Tracking.html

Oculus. (2017). *Documentation - Tracking*. Retrieved April 2017, from Oculus Developers: https://developer3.oculus.com/documentation/intro-vr/latest/concepts/bp_app_tracking/

Oculus Blog. (2013, January). *Building a Sensor for Low Latency VR*. Retrieved April 2017, from Oculus: https://www3.oculus.com/en-us/blog/building-a-sensor-for-low-latency-vr/

Popović, M. B. (2013). *Biomechanics and Robotics.* CRC Press.

RiftInfo.com. (2016, January). *Oculus Rift Specs - DK1 vs DK2 Comparison*. Retrieved April 2017, from RiftInfo: http://riftinfo.com/oculus-rift-specs-dk1-vs-dk2-comparison

Thalmic Labs (a). (2017). *Myo - Technical Specs*. Retrieved January 2017, from Myo: https://www.myo.com/techspecs

Thalmic Labs (b). (2017). *Myo - Device Listeners*. Retrieved January 2017, from Myo: https://developer.thalmic.com/docs/api_reference/platform/classmyo_1_1_device_listener.html

Thalmic Labs. (2016). *Myo*. Retrieved April 2017, from Myo: https://static.thalmic.com/sapphire/tenFootExperience/hero/myo.png

Thalmic Labs. (2016). *What gestures does the Myo armband recognize?* Retrieved April 2017, from Myo Support: https://support.getmyo.com/hc/en-us/articles/202647853-What-gestures-does-the-Myo-armband-recognize-

VR&AR Wiki. (2017, April). *Oculus Rift DK2*. Retrieved April 2017, from XinReality: https://xinreality.com/wiki/Oculus_Rift_DK2

List of Figures and Tables