Final Year Project

Immersive telepresence System v1

Prakash Waghela

2017

# Acknowledgements

# Abstract

Table of Contents

[Chapter 1 - - Introduction 4](#_Toc481247399)

[1.1 Project Summary 4](#_Toc481247400)

[1.2 Literature Review 4](#_Toc481247401)

[1.2.1 Telepresence 4](#_Toc481247402)

[1.2.2 Robotics 4](#_Toc481247403)

[1.2.3 Gesture Control 4](#_Toc481247404)

[1.3 Gap Analysis 4](#_Toc481247405)

[1.4 Technical Review 4](#_Toc481247406)

[1.4.1 Myo Armband 4](#_Toc481247407)

[1.4.2 Leap Motion 5](#_Toc481247408)

[1.4.3 Oculus Rift (DK2) 6](#_Toc481247409)

[1.4.4 Nao Robot 8](#_Toc481247410)

[Chapter 2 - Solution Design 10](#_Toc481247411)

[2.1 Aims and Objectives 10](#_Toc481247412)

[2.1.1 Functional Objectives 10](#_Toc481247413)

[2.1.2 System Objectives 10](#_Toc481247414)

[2.2 Project Plan 10](#_Toc481247415)

[2.2.1 Phase One - Input Devices 10](#_Toc481247416)

[2.2.2 Phase Two - Output Devices 11](#_Toc481247417)

[2.2.3 Phase Three - Project Deliverable 11](#_Toc481247418)

[2.3 Gantt chart 12](#_Toc481247419)

[Chapter 3 - Implementation 13](#_Toc481247420)

[3.1 Phase 1 - Input Devices 13](#_Toc481247421)

[3.1.1 Myo Armband 13](#_Toc481247422)

[3.1.2 Leap Motion 14](#_Toc481247423)

[3.1.3 Oculus Rift 14](#_Toc481247424)

[3.2 Phase 2 - Output Devices 14](#_Toc481247425)

[3.2.1 Oculus Rift 14](#_Toc481247426)

[3.2.2 Nao Robot 14](#_Toc481247427)

[3.3 Phase 3 - Project Deliverable 14](#_Toc481247428)

[Chapter 4 - Evaluation 15](#_Toc481247429)

[4.1 Testing 15](#_Toc481247430)

[Chapter 5 - Conclusion 16](#_Toc481247431)

[5.1 Future Work 16](#_Toc481247432)

[References 17](#_Toc481247433)

# - Introduction

## Project Summary

## 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

### Robotics

### Gesture Control

## Gap Analysis

## Technical Review

### Myo Armband

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



Figure - 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.

### 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 - 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.

### 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 - 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 movement 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)

### 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 - 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.
* 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).
* 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. Choregraphe also allowed the user to 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.

# Solution Design

## 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:

### 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.

### 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.

## 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++.

### 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.

### 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.

### 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.

## Gantt chart

The Gantt chart below shows a rough timescale of each phase.

Figure - A Gantt chart showing the timeline for the project

# Implementation

## 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

### 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 - 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.

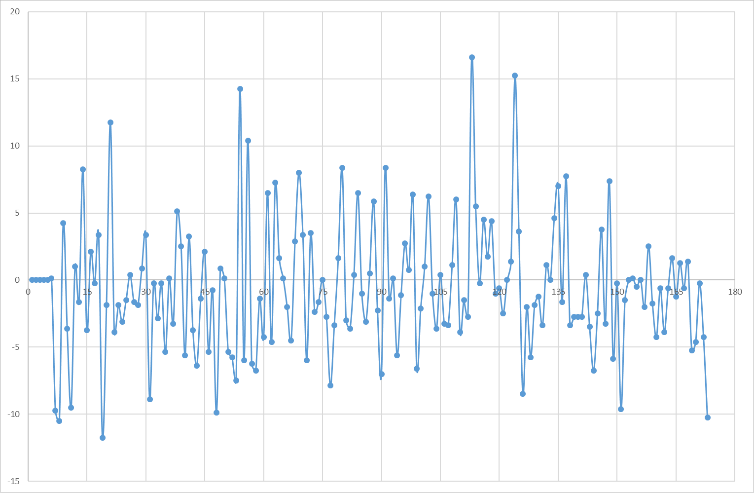
Upon completion of the tutorial, I moved on to sampling some applications available from the Myo application launcher. One such application allowed me to type on an on-screen keyboard, using my finger to point at letters and a gesture to select it. I found that I was able to point at individual letters 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.

[Image]

As shown in figure x, 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, plotting the readings from each individual sensor onto a line graph.



Despite the low level of analysis, I expected to see some form of a pattern emerging in the data. However, as shown in figure x, 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.

### 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 7, only my middle finger was perpendicular to my palm but the visualizer incorrectly detected two fingers being perpendicular.

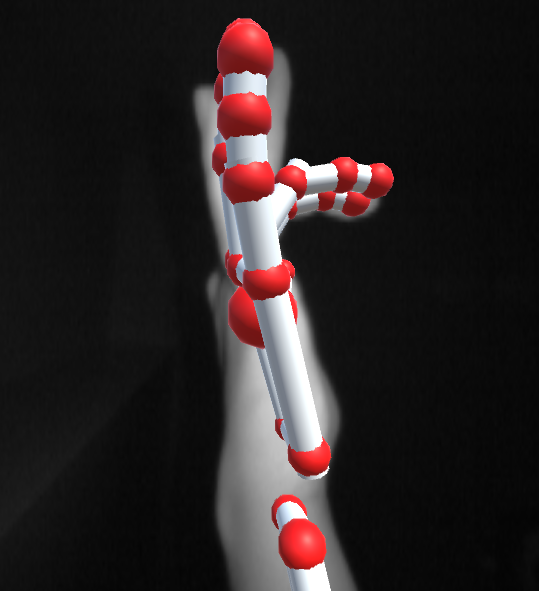


Figure - Leap Motion visualizer showing an incorrect representation

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 | C:\Users\Prakash\AppData\Local\Microsoft\Windows\INetCacheContent.Word\normal.png | C:\Users\Prakash\AppData\Local\Microsoft\Windows\INetCacheContent.Word\roll acw.png | C:\Users\Prakash\AppData\Local\Microsoft\Windows\INetCacheContent.Word\roll cw.png |
| Pitch | C:\Users\Prakash\AppData\Local\Microsoft\Windows\INetCacheContent.Word\normal.png | C:\Users\Prakash\AppData\Local\Microsoft\Windows\INetCacheContent.Word\pitch acw.png | C:\Users\Prakash\AppData\Local\Microsoft\Windows\INetCacheContent.Word\pitch cw.png |
| Yaw | C:\Users\Prakash\AppData\Local\Microsoft\Windows\INetCacheContent.Word\normal.png | C:\Users\Prakash\AppData\Local\Microsoft\Windows\INetCacheContent.Word\yaw acw.png | C:\Users\Prakash\AppData\Local\Microsoft\Windows\INetCacheContent.Word\yaw cw.png |

Table - 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.

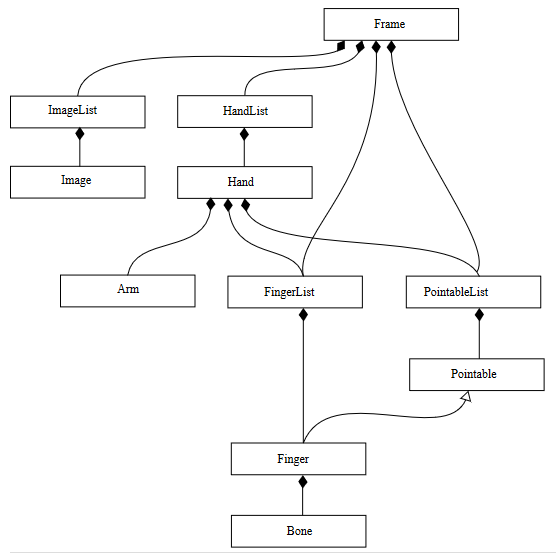


Figure - Model used by the Leap Motion to access data (Colgan, 2014)

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.

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

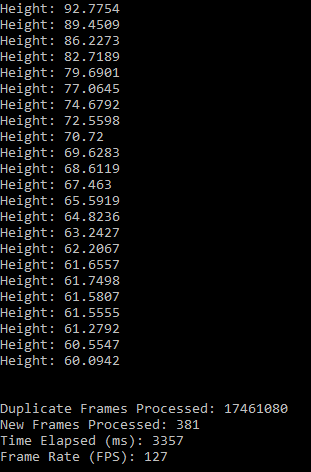


Figure - Example output from the Leap Motion application

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. Using a listener 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 am able to fine-tune how the frames would be processed, despite the increased validation to check that I’m processing the correct frame.

### Oculus Rift

## Phase 2 - Output Devices

### Oculus Rift

### Nao Robot

## Phase 3 - Project Deliverable

# Evaluation

## Testing

# Conclusion

## Future Work

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

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

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

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/

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/

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