Development of 3D Image Manipulation Software Utilizing the Microsoft Kinect

A report submitted to the School of Engineering and Energy, Murdoch University in partial fulfilment of the requirements for the degree of Bachelor of Engineering

Prepared By: Aston Ladzinski

Bachelor of Engineering

Department of Engineering and Energy

Supervisor: A/Prof Graeme Cole

Associate Supervisor: Dr. Gareth Lee

MURDOCH UNIVERSITY

PERTH, WESTERN AUSTRALIA

Abstract

Studying Engineering at Murdoch University enables students to experience many facets of engineering. In the Electrical Engineering based courses there is a vital need to have an understanding of various programming languages and methods and then explore ways in which this knowledge can be utilised. This document describes how the Microsoft Kinect can be utilized to control 3D images, specifically medical images, through the use of programming skills and software development kits.

This project involves learning how the Microsoft Kinect sensor actually works and requires the development of two programs that utilised this sensor and can be easily implemented. The first program is designed to display information that the Kinect sensor is able to detect. By developing this program, it enables a user to quickly gain an understanding of what data is available for manipulation. The second program is designed to manipulate a generic 3D image through the use of a set of gestures initiated by the user. This enables the user to see how the information retrieved from the sensor in the first program can be manipulated for useful purposes, by the creation of algorithms.

The development of these two programs required the learning of a new language. The language that was used is called C#, and it is a complex object orientated language. Through the use of the language in the Microsoft Visual Studios 2010 programming environment, it was possible to create the programs through many iterations of development. The thesis documents how this approach was made, both the method of learning C# and also incorporating and utilising the benefits of C# into the programs.

From the research conducted it was found that creating effective gestures was quite difficult as it required some form of predictive logic, and that incorporating medical 3D imagery was at a completely different level of programming skills. Despite these difficulties an effective, easy to use and modifiable program was developed that will allow future research to continue in this field. Therefore possible future projects and developments are also discussed in order to give the reader an idea of what the sensor is capable of if time and knowledge are readily available.

Acknowledgements

For their guidance, assistance and patience in completing this project, acknowledgements are required for the following people:

Project Supervisor: Associate Professor Graeme Cole

Lecturer, Murdoch University

Associate Project Supervisor: Dr. Gareth Lee

Lecturer, Murdoch University

Mr Will Stirling

Technical Officer, Murdoch University

Crystal Ladzinski and Evie Joy Ladzinski

My wife and daughter

Table of Contents

Table of Figures

List of Tables

List of Appendices

Terminology and Acronyms

API: Application programming interfaces are interfaces that allow different methods of software to effectively communicate. An API is required in order for the Kinect to be able to function in Windows.

C#: An object orientated programming language developed by Microsoft that was used to develop the programs within this project.

C++: An object orientated programming language that was intended to be used initially for the programs that were developed within this project.

CMOS: Complementary metal oxide semiconductor is the most popular method of capturing image information digitally. It uses MOSFET type transistors to be able to build logic circuits that can sense information in the form of a camera, such as is used in the Microsoft Kinect.

DICOM: A world wide convention for storing medical images in a specific format, it stands for Digital Imaging and Communications in Medicine.

HMI: A Human Machine Interface is software or hardware that allows a user to interact with a machine. This can be as simple as a button or as complex as gesture or voice recognition.

ICSE: An engineering major available at Murdoch University referred to as Industrial Computer Systems Engineering.

IDE: An integrated development environment that allows a user to develop various programs or applications with various tools, such as was used in this project in the form of Microsoft Visual Studio 2010.

LabView: A graphical based programming language developed by National Instruments specifically for the use of engineers and scientists for the initial purpose of data acquisition.

Matlab: A text structured programming language specialising in matrices and mathematics and widely used for mathematical modelling in academia and industry.

MKIMP: Microsoft Kinect Image Manipulation Program developed in this project for the purpose of manipulating 3D images.

MKIP: Microsoft Kinect Interaction Program developed for the use of understanding data produced by the Microsoft Kinect.

MRI: Magnetic Resonance Imaging is a popular form of medical imaging as it can distinguish between tissue types without the need for exposing the patient to unnecessary dangerous radiation.

NUI: Natural User Interface is an old library developed by Microsoft for the Microsoft Kinect, however recently been superseded by the more direct Microsoft.Kinect library.

Prime Sense: The Company who designed the technology behind the Kinect and created the processor that can analyse the position of a body and its comprising parts.

RGB: Refers to the Red Green Blue type camera sensor that is part of the Microsoft Kinect sensor system.

SDK: Software Development Kit used to make programming easier by allowing a programmer to utilise already available libraries, definitions or functions as is the case with the Microsoft Kinect SDK.

VGA: Video Graphics Array refers to the resolution of 640 x 480 pixels that is available with the Microsoft Kinect Cameras. It also refers to the 8-bit colour palette of 256 different distinguishable colours per pixel.

WPF: Windows Presentation Foundation is part of the .NET framework and enables user to build software by acting as a link between the user and the control item, such as what is required when programming the RGB display of the Kinect.

XAML: Extensible Application Mark-up Language is a programming language required for WPF, which was used to create a graphical user interface. It is declarative, meaning it is not procedural.

.NET: .NET or ‘dot-net’ is a framework developed by Microsoft to aid in creating object orientated programming abilities to internet applications, and is used extensively in Microsoft Visual Studios 2010.

Chapter 1: Introduction

Chapter 1 provides an introduction to the thesis. Specifically, it explains the project background and history, the scope and objective of the project, any revisions of the project and the general structure of the thesis itself.

1.1 Project Background and History

With current technology enabling us to view captured 3D images; it has become necessary to be able to manipulate these images in more natural ways. Using a computer mouse or game controller can be unintuitive and lead to frustration for the user, therefore a more intuitive method needs to be developed.

Ever since the scene in the movie "Minority Report" [1], where Tom Cruise's character manipulates images on a massive see-through screen with his hands with apparent ease, building gesture based HMI has been a priority of many scientists, computer programmers and even enthusiasts. In fact, the technology behind the Microsoft Kinect was inspired by this exact scene [1].

It is with this in mind the project was developed, so as to marry the manipulation of 3D images with a readily available sensor, the Microsoft Kinect [26]. With this technology already being utilised for applications other than gaming, the aim was to bring this technology to applications that could benefit from such technology. Medical practises would be ideal, as enabling a surgeon to be able to manipulate a 3D image whilst performing surgery would provide them with useful, accurate and in-depth information.

1.2 Project Scope

This project aims to build a program that will enable the manipulation of 3D images, using the Microsoft Kinect sensor. By utilising learned abilities in programming, C# and the Microsoft Visual Studios 2010 is required for the purpose of building this program. Through the understanding of the sensor and the programming environment, it will be possible to design features to allow this newly developed system to recognise gestures and align them with commands to interact with the 3D image.

From this initial program it will be possible for further versions of this project to be developed, looking at better methods of manipulation, and perhaps manipulation of even more complex three dimensional structures. With this acquired knowledge it would be possible to build commercial applications for whatever purpose the developer requires.

1.3 Project Objectives

The overall objective of this project is to build a software program that can recognise gestures using the Microsoft Kinect sensor, and then from these recognised gestures the HMI will allow a user to manipulate a 3D image. This objective requires the preliminary step of being able to access and manipulate the data, and therefore a preliminary objective has been established as below:

1. **Development of a Sensor Data Display and Manipulation Program**

   In order to build a program that can manipulate a 3D image, it is first necessary to obtain and manipulate data to and from the Microsoft Kinect. These functional specifications are therefore required in order to develop a management program:

   - Development of a program using C# that can be activated in windows;
   - Design of an interface that is able to be easily read by a user and display relevant data simply and efficiently;
   - Ability to obtain data from the RGB and infra-red camera on the Kinect and display the data appropriately;
   - Ability to retrieve and send data to the motor of the Kinect, and thus creating the ability to be able to change the angle of the sensor;
   - Ability to retrieve set skeleton points from the processor on the Kinect, thus enabling the ability to draw a 'skeletal' model;
   - Ability to retrieve each of the three dimensional co-ordinates from any skeletal or joint position, and be able to display this on the interface.

2. Development of a 3D Image Manipulation Program

Once the preliminary program has been developed it is then possible to design a program that enables manipulation of a 3D image, the only limitations being time, imagination and programming syntax. Well planned, functional specifications are therefore required, which include:

- Development of a program that can display a simple 3D image such as a shaded cube which can be moved using simple mouse or keyboard controls;
- Development of a number of set gestures that can be utilised to replace these mouse or keyboard controls including, rotation, movement and zoom;
- Development of algorithms that are able to be easily modified to fine tune these gestures so that they are easily used;
- Replace the mouse or keyboard controls in the 3D image manipulation program with the gestures developed earlier;
- Complete vigorous testing of this program so that it can be used robustly and by various people, in order to be ready for the presentation of the thesis;
- If time permits, additional features relevant to the project scope may be added, including:
    - Using a DICOM MRI scan image for manipulation
    - Being able to initialise a gesture to be performed
    - Adding simple voice recognition feature.

1.4 Project Revisions

Originally, the thesis project intended to build an interface that allowed 3D image manipulation, allowed a user to retrieve raw data and manipulate it via software, and also to develop a program that permitted Labview or Matlab to interface with this prior program or directly with the Kinect. As can be seen, the scope was unclear and thus needed to be modified, which was done after feedback from supervisors and completion of information gathering.

Therefore the final scope of the project has developed over time to firstly understand how to interface with the Kinect data, and then to be able to display this data in an understandable means. Secondly, the scope involves developing a software program using Microsoft Visual Studios 2010 programming environment [27], in particular with the C# [28] language, that enables a user to manipulate a 3D image (with the ultimate aim being a 3D DICOM MRI scan) using set gestures being recognised by the created software.

1.5 Thesis Structure

There are a further six chapters involved in this thesis that provide more detailed information on how the project was undertaken. The structure of the thesis is described below:

Chapter 2: Explores the technical aspects of the Microsoft Kinect, including how it functions, processes and delivers information;

Chapter 3: Introduces a reader to the language C# and Microsoft Visual Studios 2010 IDE and explains how they were utilised in the project;

Chapter 4: Describes how the original Microsoft Kinect Interface Program was designed and implemented and the final outcome;

Chapter 5: Describes how the Microsoft Kinect 3D Image Manipulation Program was designed, the considerations that needed to be made, and how the overall program was implemented;

Chapter 6: Explores the final outcome of the project, including the project itself, personal outcomes and the difficulties in completing the project;

Chapter 7: The final chapter is a summary of the entire thesis and also explores appropriate future projects.

Chapter 2: Technical Introduction to the Microsoft Kinect

The aim of this chapter is to give the readers an introduction to the Microsoft Kinect. It explains the different sensors that are combined, so as to create the overall Microsoft Kinect sensor as displayed in Figure 1. It explores how information from each sensor is processed, so that it can be utilised by a programmer or pre-developed software. It also discusses some of the limitations of the Microsoft Kinect system.

Figure 1: Microsoft Kinect Sensor [18]

2.1 Overview of the Microsoft Kinect

The Microsoft Kinect is a complex sensor that was released by Microsoft for the Xbox 360 in November of 2010 [17]. It has advanced capabilities to recognise players, gestures (if programmed), depth and voice. This makes the sensor great for controlling interactive games using a medium other than traditional game controllers or keyboard and mouse. Originally the software and hardware was developed by Prime Sense [12], and bought out by Microsoft before release. Since its release it is believed to have been sold almost 20 million units worldwide [17]. With this massive popularity, communities have developed that are interested in using the Kinect for purposes other than gaming. In June 2011 the first official Microsoft Kinect Windows SDK was released to encourage companies and hobbyists to utilise the sensor for other applications [17].

The sensor is comprised of a number of sensors that work together in order to provide useful information once it is processed by the processor. This includes: A microphone array, IR projector and camera (Depth Sensor), RGB colour camera and tilt motor with associated accelerometer sensor. The internals of the Kinect and individual sensors can be seen in the diagram below. A more thorough description of each individual sensor is conducted in the rest of this chapter. The individual sensor positions within the Kinect sensor can be seen in Figure 2.

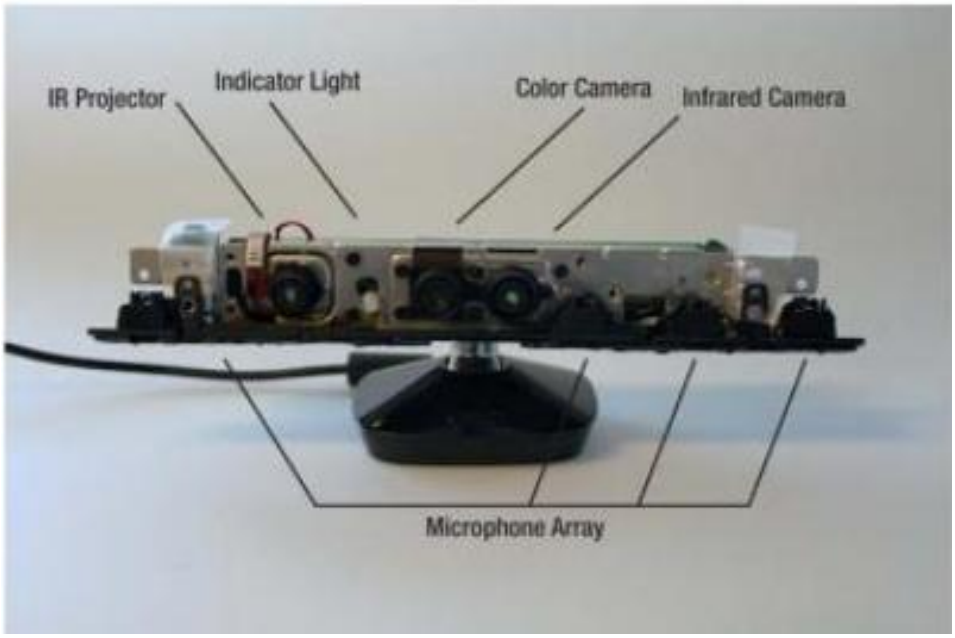

Figure 2: Kinect Component Identification with Non-Structural casing removed [1]

2.1.1 The RGB Camera

The RGB camera on the Kinect is a colour CMOS sensor used to produce an image that makes sense to the user. The refresh rate of the RGB camera is 30Hz, and if reading directly from the processor cannot be made faster, although it can be slowed down if required. The resolution of the RGB Camera is 640 x 480, otherwise known as VGA resolution and it senses each primary colour at 8-bits per pixel [17] [22].

The camera has an optimum focal length equivalent to the distance of 525 pixels, and a horizontal field of view of 62.7 degrees which is wider than the depth sensor [23]. The RGB camera is aimed at providing a method of feedback to the user; it is not typically used for the purpose of producing useful data within an application. Typically a program will not read the RGB camera data, however some new applications have software that compares depth images with colour images so as to produce facial or finger recognition. A typical RGB image produced by the Microsoft Kinect is seen below in Figure 3.

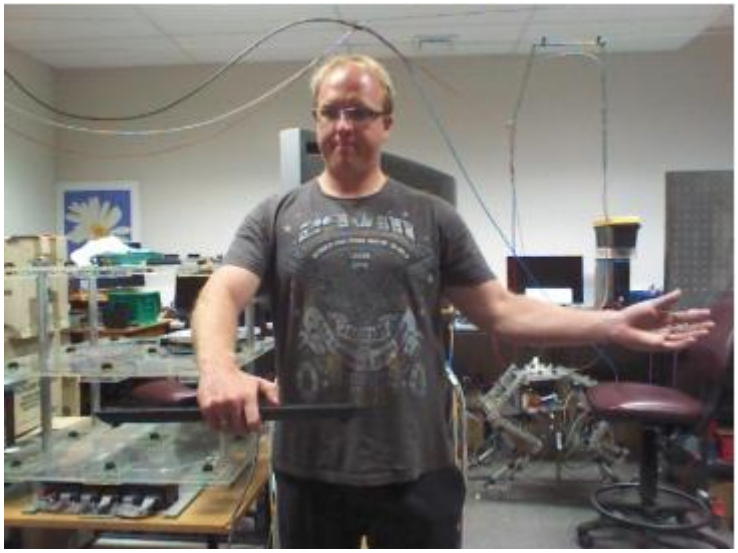

Figure 3: RGB Image produced by the Microsoft Kinect RGB Camera [22]

2.1.2 The Depth Sensor

The depth sensor is truly the defining sensor within the Microsoft Kinect, as it allows true interaction between a user and an application. The sensor is made up of two parts, an infra-red laser which projects infrared light, and a CMOS camera specifically for retrieving infra-red light information. The depth sensor camera has a resolution also of 640 x 480 pixels; however it has a higher sensitivity rating of 11-bits per pixel, giving the depth 2048 different levels of individual resolution. The depth sensor camera also reads each frame at 30Hz. The camera also has a focal length of 580 pixels, and a field of view of 57.8 degrees horizontally and 43 degrees vertically [22] [23].

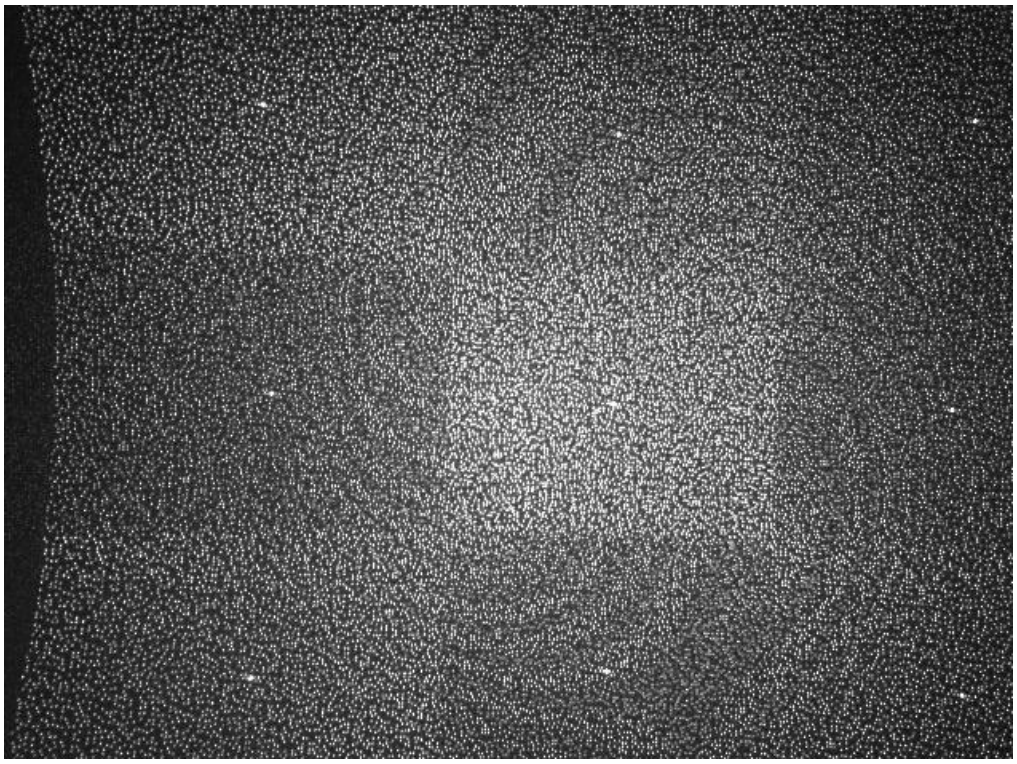

Figure 4: Image of the Infra-red light projected by the Kinect sensor [23]

The depth sensor works by shining the infra-red laser through a filter that creates a unique image as seen in the picture above in Figure 4. The infrared camera then senses each of these dots and sends them to the processor. As can be seen the dots are clumped together, and this creates unique areas of the projection so that the processer can recognise them. The processor has an image already recorded in its memory that has a known depth and therefore can compare the sensed dots to this image. The Kinect looks at groups of dots in a

9 x 9 correlation window, and compares it to other correlation windows overlapping and neighbouring the original one. From this information the Kinect is able to distinguish both the location and depth of each individual dot projected by the Kinect [6] [22] [23].

It shows that shapes can be readily distinguished through depth and shape alone, this is due to the fact that the depth sensor has a resolution of about 1cm in the Z – plane and just over 1mm resolution in the X – plane and Y – planes. This resolution is an optimal resolution at the optimum user position and will change as the user's position changes. As can be seen on the left hand side of Figure 5 there is a black bar. This bar is produced due to the windowing or correlation of the raw infra-red image once it has been processed. It should not affect most users or applications as it is not located in the centre of the image. The sensor has been developed to operate best at the distances of 1.2m to 3.5m; but it still operates if the user is 0.7m to 6m away [17] [22] [23].

Figure 5: Depth Image produced by the calculations completed by the Microsoft Kinect Depth Sensor [23]

2.1.3 The Microphone Array

The microphone array is a very important part of the Microsoft Kinect sensor, although it was not utilised in this current project. This array is made up of four separate microphones that can record at rates of up to 16 kHz in a 16-bit stream [17]. This is extremely high quality and is required for voice recognition processing by the Kinect; the sensor needs to detect not only normal voice, but also loud and soft voice volumes and frequency extremes. Below is a diagram of how the microphones are arranged. Each of the microphones streams are collated together and then passed through digital audio filters within the Windows digital sound processor in order to pass through the signal to the program or programmer [21].

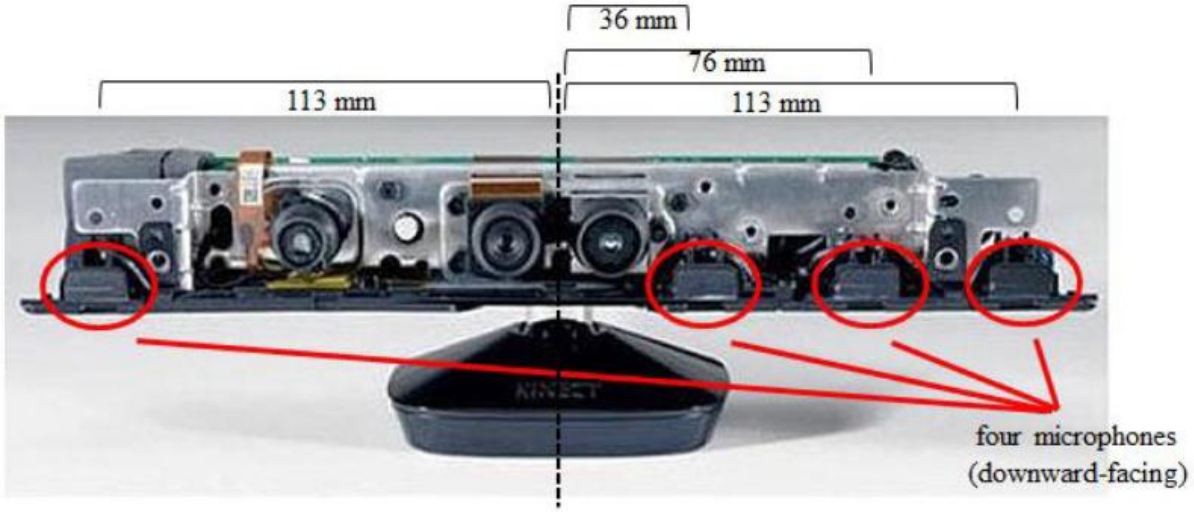

Figure 6: Microsoft Kinect diagram with Microphone array displayed [20]

Another reason for the microphones being in set locations as seen in Figure 6 is so that they can also act as sound localizers, also known as the method of 'beam forming' [20]. This is the ability of the processor and SDK to determine from where the sound originated. This can be done by comparing the sound strength at different microphones and then using triangulation methods of localisation. This could be useful in some applications, or in determining the user that spoke or made a sound when multiple users were present.

Figure 7 is a diagram that explains how the 'beam forming' localization information is produced for a programmer to use; this is in the terms of a numerical angle from the centre line of the sensor. When the user is on the right hand side of the sensor it is a negative value angle, and on the left hand side a positive angle. Ideally, the sound will be originating at a distance between 1 and 3 meters from the sensor, with an angle range of between -50° and 50°. It is understood that the localization accuracy is within centimetres of actual location [20].

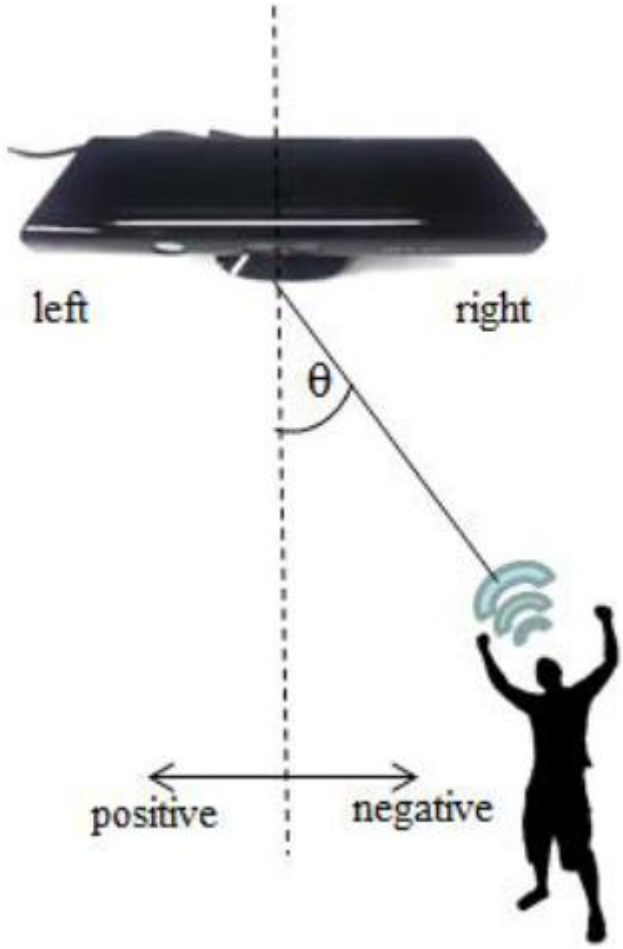

Figure 7: Diagram representing the ability of the Microsoft Kinect to localize sound [20]

2.1.4 The Tilt Motor

The tilt motor tilts the sensor up or down as its name suggests. This could possibly be used to automatically tilt up and down until the sensor can locate the user's full body. This would be a relatively simple program to implement as there is a specific tag that registers a reading of 'true' when a player is found. It would then be possible to locate all other joints as a method of initialisation.

The motor itself is seen in a picture below in Figure 8. Unfortunately it was not possible to perform a tear down, but thankfully it was completed by the website ifixit.com [18]. It is presumed the motor is a simple DC motor with a screw type gearing that drives normal gearing, in order for it to know the position of the motor it needs to have feedback. This feedback is in the form of an accelerometer that enables the processor to calculate its position and then compares this to the desired position and changes position if required [19]. The maximum range of tilt for the Kinect Motor is between 27 and -27 degrees if the sensor is set up flat. It is interesting to note, that the Kinect will register a 0 degrees only when the sensor is flat and not tilted. This means that if the sensor base is on a downwards angle then the motor must be tilted up the same angle in order for it to be registered as a 0 degree reading [19].

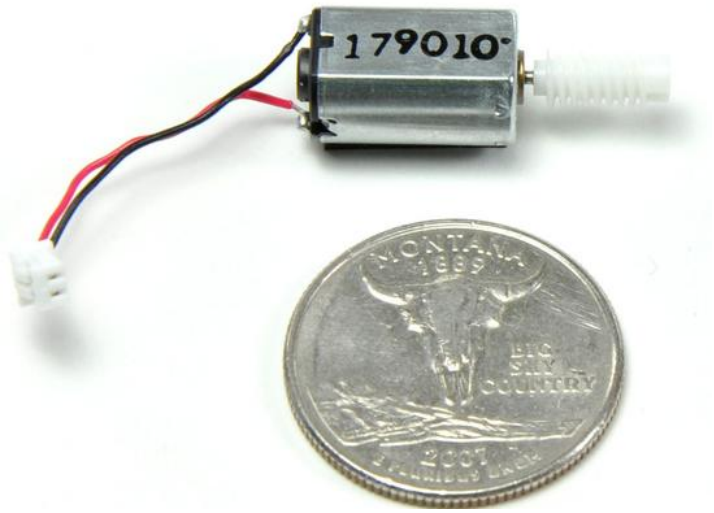

Figure 8: Kinect Tilt Motor removed from Kinect [18]

2.2 Microsoft Kinect Processor

The processor is an essential part of the whole Microsoft Kinect sensor. It was developed by Prime Sense, and is labelled PS1080 which is a fully custom integrated circuit that handles image and audio separately [12]. More detailed information about this circuit is not readily available to the public. The purpose of the processor is to receive the data from the sensors and develop useful data for the programmer or application to use. It can be seen in the Figure 9 below that it is the central component of the sensor and operates at frequency of 12 MHz from an internal clock. Once the data has been processed it creates a flag that notifies an application so that it can then request specific information from the processor [18] [24].

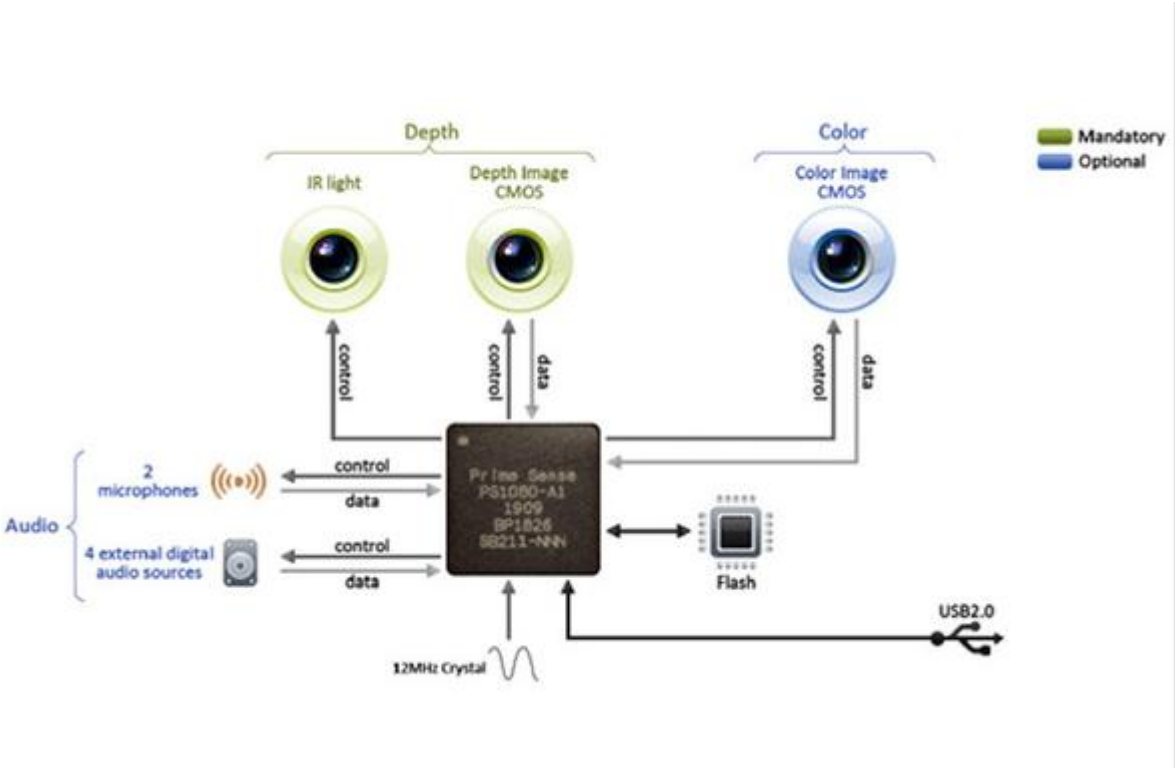

Figure 9: Kinect Processor Schematic Diagram [18]

The sort of data that is requested from the Kinect depends on the application, but the types of data that are available remain the same. The processor is able to produce raw data, such as the raw colour camera and raw depth data arrays, which are then able to be manipulated to produce images such as seen in the first program of this project. The data that is the most important for the purpose of gesture recognition is the depth data. This depth data is produced by using the infra-red camera as described earlier. The processor then collates depth data with the RGB image so that algorithms can be used to detect the shape of the body [1]. Once the shape of the body is detected from the depth data, the program can then infer the position of the joints.

This information is given in a format that specifies: Joint name (number), if the joint does exist, if it is realised or inferred, and the X, Y and Z co-ordinates of this joint. The Kinect SDK the processor is able to detect 20 joints; however this is limited to just 10 joints if the person is seated or the near sensor feature is used. Figure 10 below shows the joints that are available if the user is standing.

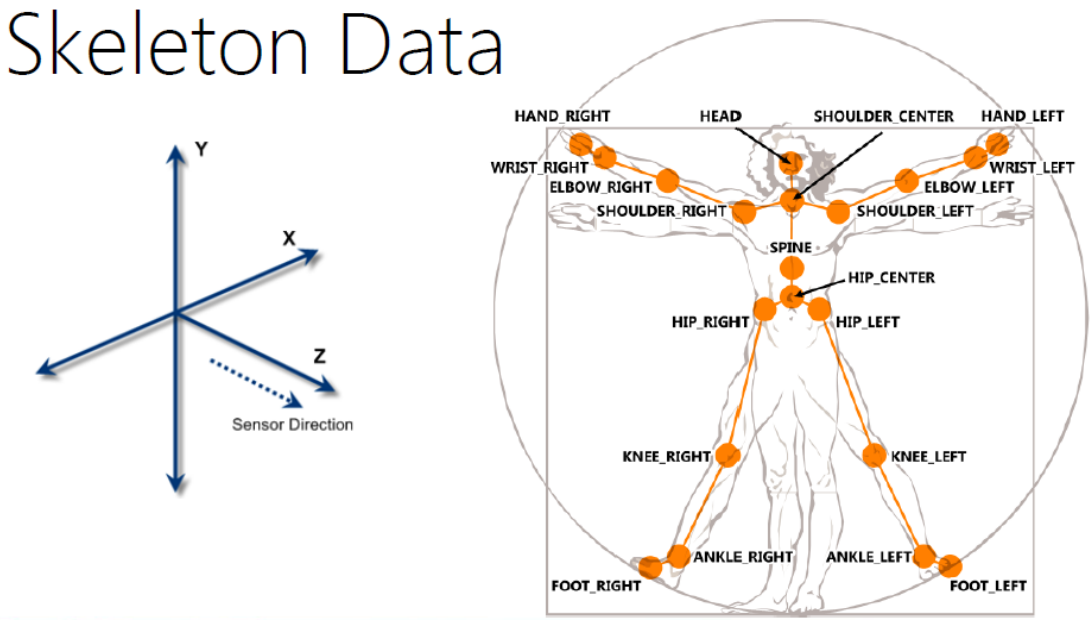

Figure 10: Kinect Skeleton Data and Joint Information [3]

2.2.1 Microsoft Kinect Communication

The Microsoft Kinect uses the USB 2.0 method of communication between the processor and either the Xbox 360 or Windows computer. The USB also delivers initialisation power to the Kinect; however the 2.5W that the USB protocol can deliver is not enough as the Kinect requires a full 12W of power supplied [22]. To overcome this, the Microsoft Kinect must be plugged into a mains power supply to supply the sufficient power of 12W. In Figure 11 below is an architecture model of the first version of the Kinect SDK; it shows how the USB protocol interacts with the software (NUI API from defunct version) and how this can then allow interaction with the applications such as is built in this project.

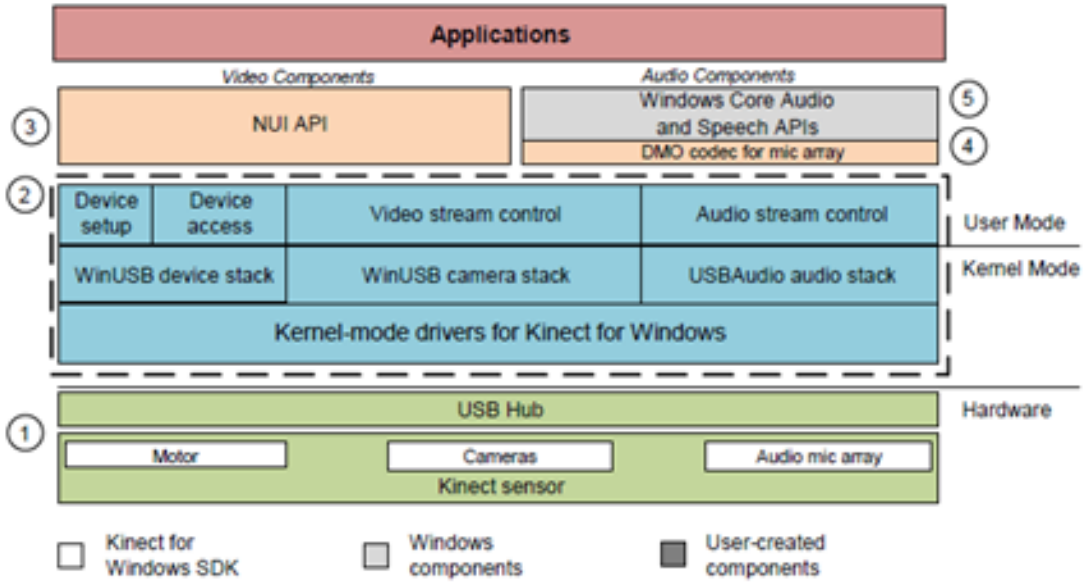

Figure 11: Kinect SDK Architecture [3]

2.3 Microsoft Kinect Limitations

As mentioned earlier the Kinect has a number of limitations for what the sensors are able to detect with regards to angles and heights. There is also a limit in detection of resolution for both the RGB camera and the depth sensor. These limitations can be exacerbated by hostile environmental situations, such as non reflective surfaces affecting infra-red mapping, infra-red sources being present that confuse the depth sensor and too many users at once [23]. Below in Figure 12 is the ideal positioning for a user of the Microsoft Kinect sensor.

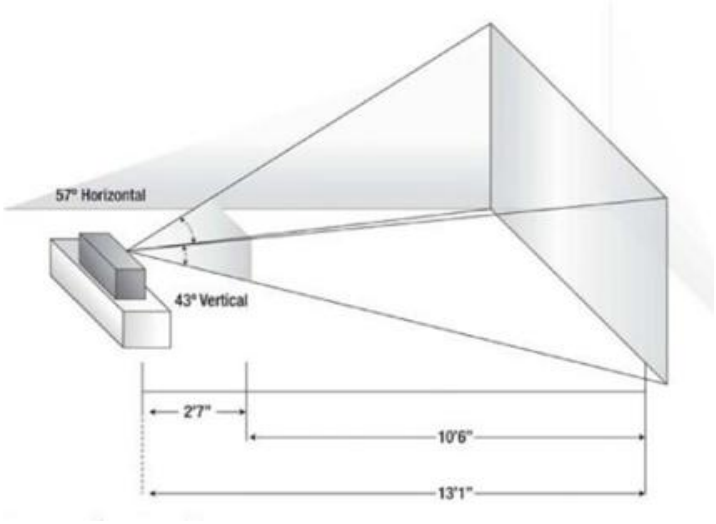

Figure 12: Kinect Sensor ideal positioning and range diagram [2]

Theoretically the sensor has the ability to recognise infinite players, providing the processor is large enough. With the current model Kinect the ability to recognise users is typically limited to two simultaneous users, however using 3rd party software it is possible to sometimes detect up to 6 users [1]. There is also an issue of not being able to pick up the joints consistently as they sometimes are 'lost'. This requires the program creator to include some method of filtering to overcome this issue [12].

The sensor also has difficulty processing sound and differentiating voice commands. Although these features were not utilised in the project, they were tested for the purpose using sample code. It was very difficult for the Kinect to be able to pick up even pre-defined words such as 'yes' or 'no', perhaps due to an Australian accent.

Chapter 3: Introduction to C# and Microsoft Visual Studios 2010

This chapter has the purpose of introducing to the readers the C# programming language including the history and an overall look at the general structure that was used for the Kinect. It also discusses why it was used for the purpose of this project over other languages. Furthermore the chapter discusses the use of the programming environment Microsoft Visual Studios 2010, giving a brief overview of its features and how it was implemented in the project to create an interface.

3.1 Overview of C#

Originally for this project it was intended to learn C++[30] as many of the demonstrations developed for the Microsoft Kinect SDK were written in this language [3][4]. Many hours were spent to learn the syntax and structure of C++ and trialling sample code. It was found that after a period of time learning about it and with the consideration of restricted time frames that C++ might not be the best language to learn for this project. This is because C# is created by windows and uses included framework, unlike C++. Therefore it was left to choose between other related programming languages that are compatible with this SDK such as C, Visual Basic and C#. After some extensive researching C# was chosen, because it was the newest and easiest to use, compared to C, which is a relatively old language and Visual Basic, which, although having been used and studied, may be limited in the control over the Microsoft Kinect and its advanced capabilities.

Even though the Java language has many common features with the C# language it was not chosen or even considered for a primary reason. That reason is that C# is a programming language that was developed for release in 2002 by Microsoft, and its latest iteration is C# 5.0 covered under ISO 23270:2006 [31] [2]. For interest sake a comparison between Java and C# can be found at reference [32]. Its name is pronounced, ‘C Sharp’ and not ‘C Hash’ as some people commonly call it. It was named Sharp because in music the sharp symbol (#) refers to the note being one semitone higher, this is in reference to C# heritage. C# is derived from the programming language C which was developed at Bell Laboratories USA in 1970 [2]. In 1980, Bell Laboratories USA also released a new more advanced style of

programming known as C++ or C “Plus-Plus”. This name is derived from the fact that in the C programming language, by coding and ending in the term ‘++’ it would add one to the item – in other words C++ is one up from C. In this new language C++, it enabled the continual reuse of newly created code, this type of programming became known as object-oriented programming [2].

The reasons C# was released was that it permitted faster and more intuitive learning of a C based language compared to C++, allowing it to be more accessible. C# also had more features than C++, creating more functionality. C# was released as part of the .NET framework. .NET is pronounced ‘dot net’ and its purpose was to enable programmers to create interactive websites by using components, similar to how C# utilises object orientated programming [2].

3.1.1 Learning of C#

It is outside of the scope of this report to detail exactly how to use C#, although it was the language that was used for this project; a C# tutorial is not the purpose of the project. Instead this section explains briefly how the language was learnt and the basic structure of it. As mentioned earlier, C++ was initially the language that was to be used in the project and much time was spent learning this. Fortunately not all of this time was wasted, as structurally C# is very similar to C++ as it was derived from this language. Also due to previous experience in some Murdoch University Engineering units, previous exposure to Matlab, Labview, Basic and Forth was established. This foundation enabled C# basics to be learnt at a fast rate, considering the short time frame of 15 weeks to learn and implement it into the program [14].

For the purpose of learning C#, firstly a compiler needed to be installed on the computer. This was organised by Will Stirling, whom installed Microsoft Visual Studio 2010 [27] onto a number of computers in the Mechatronics Laboratory. Once this was completed a number of sample programs containing the code that came with the Microsoft Kinect SDK were trialled and run. Some of the code did not function and will be discussed in the Project

Difficulties section. However the ones that did work enabled further study of the basic structure of the code.

Once the C# language was sufficiently understood learning shifted focus to building non Kinect related programs, such as simple counting programs, iteration programs, if-else statement programs etc. What was built was essentially all the basic structures of any general programming language. These programs were built with the aid of three main reference texts including: C# for Students [2], C# Station - C# Station Tutorial [8] and Data Structures and Algorithms Using C# [13].

Initially confusing, it soon became apparent that C# is relatively straight forward to use and makes programming quite intuitive. It is has the ability to reference various libraries that have been created by Microsoft or 3rd parties. These libraries contain many functions that can be utilised, from basic mathematical functions to advanced 3D image rendering functions. It is also possible to create a user's own functions or methods as they are called in C#, which can be recalled at any time [2]. Another useful feature with C# is enabling variables or functions to be either public or private [2]. By allowing them to be public they can be seen by anywhere, but by being private the variables can only be used within that function. This enables variables to be named without worrying that it has been used somewhere before if the function is set to private.

3.1.2 Utilisation of C# for Microsoft Kinect

In order to understand exactly how C# could be used to program the Microsoft Kinect, a number of sample programs that came with the Microsoft Kinect SDK were examined. These sample programs were quite advanced to understand as a beginner to C# language, and therefore some elements were not recreated in the final programs. However the parts able to be understood formed some of the basis of the final code in the following programs. This includes being able to display visual data from the Kinect in different formats.

The initial code helped to understand that each of the different sensors, or processed data points such as skeleton image streams, are able to create a flag or trigger an event each time they change. This flags occur when a certain condition has been met and lets other

methods in the program, which are looking for this flag, run. Triggers cause another method to run when a condition has been met. Therefore it was understood its best to look at when these events change, such as the image that the RGB camera took every 1/30 of second. By then performing a task such as displaying this image in a box, it would be possible to have a video feed. Similarly, when the skeleton point is processed by the Prime Sense chip analysing a specific element of data, it creates a flag that can then notify the programmer that new information is available. By making use of this information, through creating programming algorithms, it is then possible to create gesture recognition and image manipulation programs. This can be as simple as looking at certain velocities of a particular joint, or as complex as combining acceleration, velocity and position of multiple joints with filters.

3.2 Overview of Microsoft Visual Studios 2010

Microsoft released their integrated development environment (IDE), Microsoft Visual Studios 2010 in April of 2010 [15]. Its purpose is to create an environment that allows a programmer to program in various languages to build applications for the Windows operating systems or internet based interfaces. It has a number of areas that enable full construction of an application, including a code editor, a debugger and a design area.

The code editor allows a user to create a program in various languages. The software comes with a number of languages including C++, C# and Visual Basic. Other languages such as Ruby and Python are available for use via plug-ins that can be purchased or downloaded. The editor allows the ability to comment sections and of course minimize sections of code. It also allows a user to easily switch between programs through well displayed library and program trees [15] [16].

The debugger, as its name suggests, allows a user to debug the code that has been built. The debugger is highly advanced as it is not only a source-level debugger, it can also operate as a machine-level debugger. It has other functions such as breakpoints, step in and step outs, all of which may be useful in building a program and debugging it efficiently [15] [16].

The design area is unique in that it allows a programmer to develop the interface separately from the actual program running in the background. It uses a number of tools to increase productivity including Windows Form Designer, WPF Designer Mapping designer and more. These tools enable programmers to drag and drop common structures such as buttons or text boxes directly onto the display panel and code will automatically be created in the designer code window. This code can be further modified using the XAML language to enable different functions such as naming the button, linking the button to variables or other functions [15] [16].

Overall, Microsoft Visual Studios 2010 offers a complete package to create applications and programs in the Microsoft Windows operating system environment. With various languages to utilise, the system allows programmers from different language backgrounds to use a common and relatively simple IDE [16].

3.2.1 Learning of Microsoft Visual Studios 2010

In order to learn how to use Microsoft Visual Studios 2010 an extensive range of YouTube.com videos were viewed. Also the book C# for Students [2], which is endorsed by Microsoft, has a general outline on how to use an older version of the software, Microsoft Visual C# .NET. This software is very similar to the current version and aids in creating an understanding of the general layout of the program.

Once this information had been digested a number of the sample Microsoft Kinect SDK programs were trialled so as to get a better understanding of how the system works. In particular the previously discussed design area was a new method of programming, both syntactically and concept wise. Through continued use and trial and error of different functions within this environment it was possible to receive a good foundation in the Microsoft Visual Studios 2010 IDE. There is still a vast amount of information to learn about this program, and many features could be better utilised if time was available. However, simply, it was learnt how to be able to edit, compile, debug and run programs in the environment and thus make it possible to continue with the project.

3.2.2 Incorporating C# in Microsoft Visual Studios 2010

The final stage required to start programming the Microsoft Kinect, is learning to be able to utilise C# in the Microsoft Visual Studios 2010 IDE. The method for doing this is creating a project within the environment that is orientated in the C# language. Once the project has been created it is possible to start programming in the C# environment. However, it is necessary to first add libraries to the reference list. By adding these libraries to the reference list, it is then possible to refer to these libraries and utilise functions that they have incorporated.

These libraries have to be activated, and the method of doing this is documented in the code that is attached in the appendices. Apart from the libraries the code will also occasionally make reference to the HMI area of the IDE. The names of whatever are being referred to are called tags, and need to be carefully selected. The C# code can change the value of the front item that is in XAML code, but also read from the XAML code using C# commands in the program. It is essential that good coding practises are maintained so as to

minimise any cross contamination that could occur by accidently having same named tags of items on the HMI and variables.

Chapter 4: Development of the Microsoft Kinect Interface Program

This chapter describes how the original interface program was developed. It discusses the original ideas, how these ideas were implemented and the final outcome of the program.

4.1 Original Idea

In order to even start considering how to build the Image Manipulation program, it was essential that a program was designed enabling interaction with the Kinect. It is vital to build a program that allows a user to easily see the data that is generated by the Kinect. Without knowing how to obtain this data, or understanding how to manipulate the data, it would be impossible to continue further with the thesis.

With the original idea of this program, the main objective was to create an interface that simultaneously displayed all different video information. This included having a stream of the data that could be used to display an RGB image, a stream that could display the depth data and also a dynamic image that could display the approximate positioning of the skeleton and its joints.

Other objectives within this original idea included the ability to display individual joint locations if required. This meant that the spatial location of any joint recognised by the sensor could be displayed numerically by allowing the X, Y and Z co-ordinates to be retrieved. The tilt motor was also incorporated into the original panel so that it could be manipulated if required. This would possibly be required if the sensor needed to track or locate a person in order to get the user in the most ideal location for recognising gestures.

Finally, it was intended to display the microphone or sound information in the form of a dynamic sound wave graphic. This would simply show the frequencies and amplitude of the sound energy that the microphone received. However on top of this graphic, the aim was to include a directional arrow that would point to the direction that the sound was originating from. This is possible by utilising the microphone array sensors in the Microsoft Kinect that can differentiate between directions of origin by comparing the sound energy strength and triangulating these results.

4.2 The Development Stages

There were several minor objectives that needed to be accomplished in order to complete this particular part of the project. It was also necessary to move forwards in the project or it would not be possible to properly build an image manipulation program using the Microsoft Kinect. The objectives were as follows:

- Display RGB Camera Data
- Display Depth Sensor Data
- Display 'Skeletal' Tracking information
- Manipulate the Kinect Motor
- Determine the co-ordinates of individual joints
- Display all of this information on a easy to understand interaction panel
- Display sound data and sound localization graphics

Each of these objectives needed to be realised in order to develop the MKIP. Fortunately there were many sample programs available in the Microsoft Kinect SDK, and these programs created a basis for the MKIP. By testing these programs and checking how each of them was coded, it was possible to reuse and recreate the code that would allow the objectives above to be achieved. Each of the sample programs examined was a necessary component of the MKIP, so the aim was to create a program that could include all of these elements in one simple program.

The difficulties with the development of this program were that the programming language and syntax needed to be learnt quickly, whilst also utilising a new interface (Microsoft Visual Studio 2010). Therefore to start learning methods of developing the MKIP, the sample programs were iteratively modified to see the effect that different code had on the program. Some of the lines of code were even deleted to see if it did have any real effect on the program behaviour. From this trial and error method of learning, an idea of how the sensor interacts with the application and programming language was developed.

The RGB Camera data was reasonably easy to develop as it directly reads the image to a stored pixel matrix. This image is written over each time a colour camera event changes, which is at a rate of 30Hz. This information is important to display when the user needs

some sort of feedback for their position. This data is then able to be read on the user interface in the form of a video by using the XAML language to produce an image that continuously updates from a specified address.

The depth data was more difficult to obtain as it is not in a normal data format that can be written directly to an image. In fact there is 2 bytes of the data per pixel which can be seen in Figure 13, 3 bits of which are player information and the remainder depth information. When the depth frame ready event has been flagged, then the program needs to copy this information to an array that can be manipulated.

In order for the depth information to make sense to a user visually, firstly the player index information needs to be removed; this can be done by using the short command to shorten length. Once this is complete the information needs to be converted to a RGB image, of a certain shade of colour. This is achieved by removing the blue, red and green parts of the depth information. Finally the depth data that is classified as unknown or too far is mapped as black so as to distinguish between known and unknown depth data. Once this is completed, similar to the RGB Camera image, it is written into an image buffer continuously at 30Hz.

Figure 3-3. Layout of the depth bits

Figure 13: Layout of the depth bits received from the Microsoft Kinect [1]

The skeletal data, or joint information, is received from the Microsoft Kinect SDK in the form of an array. This array needed to be renamed so as to identify each joint individually. Extensive testing was conducted to find out what each joint was labelled. Once this was known, through trial and error, specific points had lines drawn between them, such as the hand and elbow, so as to create the appearance of a skeleton. This skeletal information is then written as an image; just like the depth and RGB camera data every time the skeleton ready event is flagged which causes a method to run that retrieves this image.

The skeleton information is not useful except as user feedback, and therefore it is desired to be able to detect individual joints for use in the program. This was achieved by breaking down the array of that was created by the joint itself. Within this array is located the actual X, Y and Z co-ordinates of the joint processed by the Kinect and the Microsoft Kinect sensor. This information is displayed on the front panel by associating it with a label created in XAML in the design view of the IDE. The method of extracting and displaying this data was aided through the use of the C# for Students reference [2] and also many attempts at trial and error.

A motor is within the sensor and moves the body of the Kinect up and down. This is used in detecting users in the camera's field of view. It moves rather slowly and is not intended to be used frequently. It was decided to include this in the MKIP so that some sort of output to the sensor was programmed rather than the usual data retrieval. This was achieved using the XAML language that enables a user to place a slider at the desired camera angle. The value is then written to the Kinect so that the angle is changed when the 'Set Kinect Angle' button is pressed and causes the event to flag.

The difficulty with this part of the program is measuring the value of the sensor's tilt position when the program is started. This is because when the command should be initiated was not known, and the places it was called at that seemed logical did not work. It was eventually found through trial and error, that as soon as the sensor was initiated, the value could be read once and displayed. Issues persisted, however, as the value seems to be a couple of degrees out if the sensor is set to 0, and then the program restarted. This issue could not be resolved, though it seems that it should be straight forward as the accelerometer should detect actual position.

Redesigning the panel was essential so that it is easy for a user to interpret. In the original panel before it was cleaned up, there was data everywhere, some of which was redundant. This was cleared up by removing the non-essential data. Instead a limited selection of viewable data was added to the panel. This information gives the general idea of the panel in an easy to understand layout. It is hoped that, from this panel, future users could also develop their own programs more easily, by understanding what data the Microsoft Kinect is capable of delivering, except for sound sensor information.

4.3 The Project Outcome

The original idea was to produce a program that would enable users to view all the Kinect data in an easy to understand and meaningful way. This was not entirely achieved as there were difficulties in reproducing code, or producing new code that enabled the captured sound activity to be displayed in a purposeful way. Since it had little relevance to the entire outcome of the main objectives of the project it was placed on the list of things to do if there was extra time or trial in future projects.

As can be seen in Figure 14 below, the final panel shows a number of interesting data streams that will be useful in the Kinect Image Manipulation program. Each of the sections will be explained in order to make sense to the reader that is unable to see this program in action.

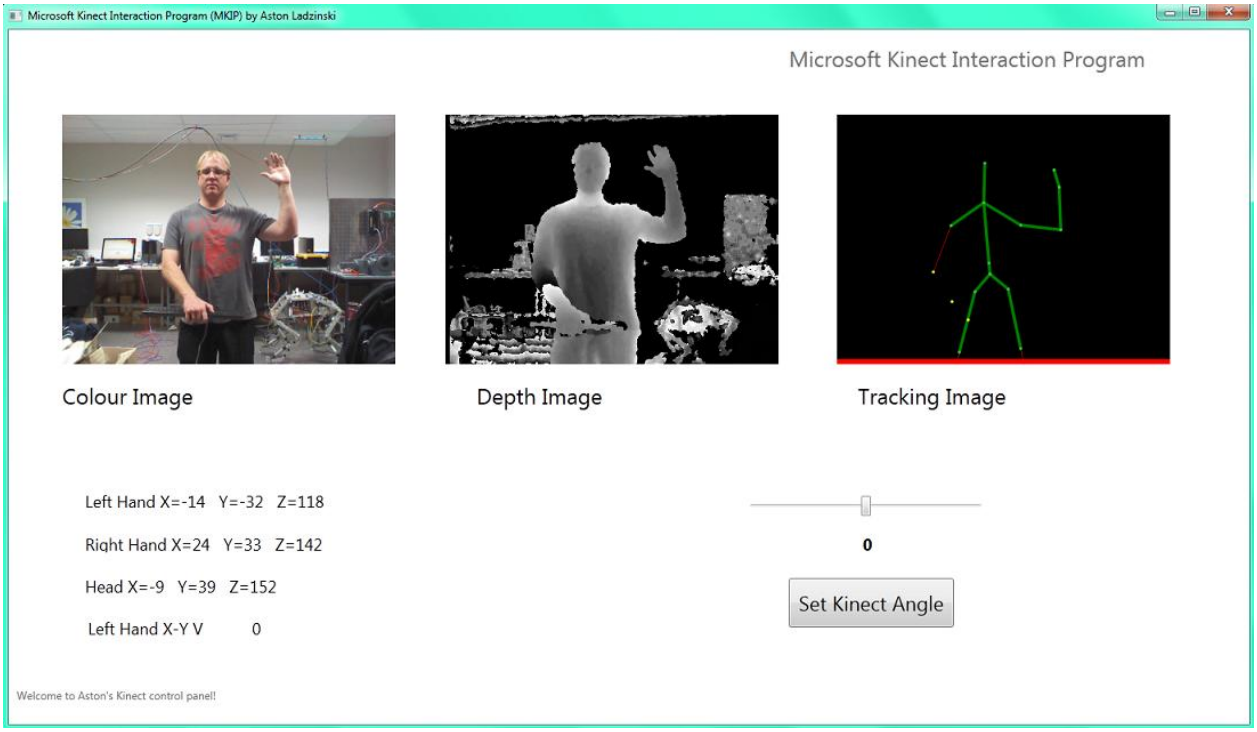

Figure 14: Kinect Data and Display Manipulation Program

On the left hand side of Figure 14 the image that is displayed is the RGB colour image as recorded via the RGB Camera sensor of the Microsoft Kinect. This image changes at 30Hz and is a vital piece of information for a user as they are able to see an image that they can easily understand, hence providing visual feedback.

The middle image is the depth image, and also has a rate of 30Hz. It is not overly useful for the direct user as humans do not produce depth imaging using the same method. However it was seen that as the user moves back and forwards or as an object is moved within the view, then the image changes. There are various levels of depth that change in intensity of gray scaled pixels as the depth changes. This image is more useful for the programmer who wants to understand the depth of each pixel for the purpose of gesture recognition.

The image displayed on the right aimed at representing the 'skeleton' of the user. This is not meant to be an actual skeleton, rather it has lines drawn between points that the sensor is able to recognise and that make visual sense, i.e. left shoulder and left elbow. This image is also run at 30Hz as it relies on the depth image to be processed. If a joint is not known and is inferred then the joint is coloured yellow and the line between joints is coloured red. This image is great for understanding what information the Kinect has available for building algorithms for recognising gestures and also seeing the accuracy of the joint processing.

In a box on the left hand side is the sample of joint information that has been retrieved from the Microsoft Kinect. This information is in the form of X, Y and Z co-ordinates and is in the unit of centimetres. Typically the Kinect information is available in meters; however this has been converted so as to be able to detect smaller changes visually. The co-ordinate information is what will be actually utilised for the gesture recognition.

Finally on the right hand side is a slider and 'Set Kinect Angle' button. These features allow a user to select the angle of the Kinect via the slider and then press the button to set the angle. This feature could be used to automatically track users if there was an issue of the Kinect being at the wrong angle and unable to detect any users.

Overall this program, which is vital to being able to continue on with the rest of the thesis, was a success. Even though the sound component of the program was not realised due to its difficulty and excessive time consumption, its inclusion was deemed unnecessary to complete the next section of the thesis. The display is easy to understand and the program is very robust, and apart from the ability to control the motor, the rest of the functionality relies on only retrieving data. This part of the project took a little longer than expected due to learning the languages involved in developing the program, but it was required to be completed before other sections could be progressed.

Chapter 5: Development of the Microsoft Kinect Image Manipulation Program

This chapter looks at utilising the information from the previous chapter to create a program that enables a 3D image to be manipulated through the use of gestures captured by the Microsoft Kinect. It considers the original idea, the requirements of the overall program, how the program was developed, and the final stage of the program at the time of compiling this report.

5.1 Original Idea

As discussed earlier, the original idea came from the movie "The Minority Report" [1], where a character is able to interact easily with a computer via gestures. Therefore it was decided to utilise the new Microsoft Kinect SDK for Windows to try and replicate some of these features in the real world.

Originally the whole premise was to be able to use these gestures to manipulate a 3D medical image, as this seems like a useful feature for medical applications. In the medical field 3D imagery is becoming much more common, as the technology behind capturing these images improves. The improvement is due to the increase in sensor capability, new techniques for image capture and of course the means to store massive amounts of data electronically in readily accessible databases [1]. However being able to manipulating this imagery often requires the traditional mouse and keyboard.

Being able to only manipulate the 3D image via a mouse or keyboard is unintuitive and also limits the applications of manipulating the image. Currently in surgical operating theatres static images are displayed such as X-ray films or printed MRI sectional scans. These images are useful, however they are not utilising the full potential of the image if they are recorded in three dimensions. Due to strict sterilisation procedures, using traditional methods of interacting with a computer system within the surgical room is just not feasible. There needs to be a method to control a 3D medical image without causing a breach of sterilisation, and that is intuitive and simple to use.

By using gestures based recognition, it is proposed that a surgeon or physician's aid will be able to manipulate 3D imagery on a screen within easy view. This will help the physician to receive the most detailed information whilst performing an operation, without having sterilisation issues. The 3D image information is more detailed as it can be changed dynamically and also be rotated, shifted or zoomed in and out. This practise of providing more information can lead to better patient care, a much desired outcome in medical practises. There is also the possibility of adding voice recognition to aid in controlling the image, so as to free the surgeon's or aid's hands.

It was decided to concentrate on a more simplistic 3D imagery that is cubical in shape. The reason a DICOM image was not used was due to the complication in producing code that would be able to call up parts of the image relevant to the desired view to give 3D effect. This was concluded so that the project would be completed on time, and if it was completed ahead of schedule then the option to include medical imagery was the next step of the project. Utilising the previously developed Microsoft Kinect Interface Program, a number of gestures were decided upon and incorporated in order to be used to manipulate this image. Following this section of the report is the specific requirements of the gestures and the program, and how this program was actually developed and its final outcome.

5.2 Specific Requirements

In order to complete this project and make it functional and intuitive, a number of requirements were set and objectives created. In particular the gesture commands needed to be thoroughly considered, not just what they were intended to control but also how they were composed and many other factors. It is also essential to understand what is required in an image that is to be manipulated and how it's possible to choose something appropriate for a demonstration purpose.

In order for the program to functionally manipulate an image there needs to be a number of different gestures developed. Each of these gestures will be unique so that they are recognised by firstly the processor (spatial location) and then directly by algorithms developed in C# coding. Each of the gestures may however share similarities with other gestures. This is for the purpose of establishing both an intuitive feel to the gestures, and so that the amount of different commands that need to be performed are limited. The actions required to be performed by the gestures are included in Table 1:

Move Left	Move the image being manipulated to the Left
Move Right	Move the image being manipulated to the Right
Move Up	Move the image being manipulated Up
Move Down	Move the image being manipulated Down
Rotate Left	Rotate the image being manipulated Left about its axis
Rotate Right	Rotate the image being manipulated Right about its axis
Rotate Up	Rotate the image being manipulated Up about its axis
Rotate Down	Rotate the image being manipulated Down about its axis
Zoom In	Zoom In on the image or making the image larger
Zoom Out	Zoom Out from the image or making the image smaller

Table 1: Required Gesture Types for Kinect Image Manipulation Program

Once these actions were confirmed it was then necessary to consider the actual requirements for the gestures so that they were enabled. It means that various areas of the physical manipulation needed to be considered in order for it to become a gesture. Some of these considerations include:

- The reliability of the gestures
- The robustness of the gestures
- The ability of the Kinect to distinguish between Joints
- The ability of the Kinect to distinguish small movements
- How to initiate a gesture
- How to ensure gestures are not confused or too similar
- How to create a gesture that was intuitive
- Being able to routinely perform the gesture without injury or pain
- The size or scope of the gesture
- How capable the Kinect is at distinguishing velocity

Each of these areas were considered in order to make the best possible gestures with the goal in mind of being able to present a working program at the end of the thesis for the presentation day. Some of the considerations were merely thought of; however others were involved extensively in the developmental stage and will be discussed in subsequent areas of this report.

After numerous tests involving various gestures that shall be discussed in the next section of the report, a final and simplistic number of gestures were chosen. The gestures that were finally decided upon in Table 2, below:

Move Left	Velocity of greater than 3cm/s using left hand in left direction whilst hand is positioned 3cm left of elbow X position
Move Right	Velocity of greater than 3cm/s using left hand in right direction whilst hand is positioned 3cm right of elbow X position
Move Up	Velocity of greater than 3cm/s using left hand in upwards direction whilst hand is positioned 3cm above elbow Y position
Move Down	Velocity of greater than 3cm/s using left hand in downwards direction whilst hand is positioned 3cm below elbow Y position
Rotate Left	Velocity of greater than 3cm/s using right hand in left direction whilst hand is positioned 3cm left of elbow X position
Rotate Right	Velocity of greater than 3cm/s using right hand in right direction whilst hand is positioned 3cm right of elbow X position
Rotate Up	Velocity of greater than 3cm/s using right hand in upwards direction whilst hand is positioned 3cm above elbow Y position
Rotate Down	Velocity of greater than 3cm/s using right hand in downwards direction whilst hand is positioned 3cm below elbow Y position
Zoom In	Move Head approximately 3cm to the Left of the centre of the Spine
Zoom Out	Move Head approximately 3cm to the Right of the centre of the Spine

Table 2: Final Gesture Types for Kinect Image Manipulation Program

Many different rotate and move gestures were initially considered and trialled and these will be discussed further in the development section. The gestures specified in the above Table 2 were chosen due to their ease in programming and their reliability to be detected by the Kinect. They are also intuitive as one arm is for the purpose of manipulating the position of the object within space, and the other arm is used for the purpose of manipulating the object about its axis. A user simply waves their hand in a specific direction at a high enough velocity in order for the object to move or rotate. The direction that their hand moves is the direction the object should move or rotate, however the movement also needs to be on the same side of the elbow as the movement direction. This intends to remove the issue of the object returning to its initial origin if the hand was moved first left and then right from a central location. Both hands can be recognised simultaneously and conduct their action.

The Zoom In and Zoom Out gestures may seem unusual, as using the head is not a typical means of manipulating something intuitively. However, often people use their head to change their point of view to enable them a better view. More importantly, it was found that the Kinect was able to detect small movements of the head very accurately. This is for two main reasons: One is that the head is typically stable on people as it aids us in our natural balance and also we require a stable head in order for our vision to remain accurate and effective. The second reason is that the head is located far from other joints, and there is little likelihood of something passing occluding the head due to our natural biomechanical dynamics. In fact, if our head is not being recognised and the Kinect sensor is located in a sensible and useable space it is unlikely that the user would be able to interact with whatever program is being used as an interface.

Having gestures chosen, it was now necessary to consider what image should be used in the manipulation program. An image that was easy to manipulate was required due to the complexity of building a 3D image from a DICOM MRI scan. This would require not only translating the image into a useable format, but then also using complex mathematical algorithms to point to each part of the picture to form a 3D image. Therefore a simple image was chosen, in particular a cubic trapezoid in the colour blue with a black background. This can easily be changed to any colours however it was chosen purely for ease of sight and aesthetics. This image needed to be controlled via the gestures; however this was the final stage. First, for testing purposes, the image manipulation was completed with the use of a mouse.

5.3 The Development Stages

Objectives for the Kinect Manipulation Program included:

- Be able to display or locate individual co-ordinate sections (i.e. X co-ordinate only)
- Have the ability to determine velocity of a joint
- Build simple gestures that can be recognised and change a Boolean value on change
- Build a display that shows gestures are recognised via changing lights
- Build a display of a 3D image that can be manipulated
- Combine the manipulation of the 3D image with programmed gestures
- Build a filter to attempt to eliminate false positive of the gestures

The first objective that was targeted was building a filter for the joints, as it was noticed that joints were continuously changing even if the joint of the user seemed to be stationary. It was thought that some sort of filtering mechanism would perhaps solve this issue. In order to do this a 5 point moving average filter was built for each points X, Y and Z co-ordinate. Building this filter did stop the joint changing more than necessary, however there was still movement of the joint perhaps due to the changing depth pattern or minute movements of the user.

The next objective to work on was developing an algorithm that was able to detect the velocity of a joint. In theory this is not difficult as it is the derivative with the respect to time. It was assumed that the distances would be in a single plane, and that each unit of time would be 1/30 seconds due to the 30Hz refresh rate of the cameras. After many weeks of continuously varying velocity, and many issues combining the velocity with gestures, it was found out that the depth data is NOT refreshed at a rate of 30Hz. In fact it was discovered from a post on the stackoverflow.com forums, that the latency of the joint co-ordinates was in fact closer to 100ms. Once this was changed, the velocity, and thus the gestures became a lot more accurate and responsive.

Developing the gestures required many hours of trial and error. It was necessary to first create the gestures that were very simple to be recognised, such as the left arm being placed across the centre line of the body or the arm being held above the head. These gestures were large in movement and therefore easy to recognise, however these gestures are unrealistic in a surgical setting and therefore it was necessary to improve on them. By utilising the ability to detect the velocity of the joint as well as the location, it was possible to build algorithms that recognised gestures. Although the gestures used are still quite basic, they enable a user or new programmer to see how they can be modified to become more complex and thus recognise more intuitive gestures.

Of course without being able to tell if these gestures were being recognised or not, it would be irrelevant to work on developing better gestures. It was required that a simple method of knowing if a gesture had been initiated was developed. The method that was used was creating a light on the front panel that would change from red to green in colour when the gesture was activated. In order to do this a Boolean value was changed from true to false when a gesture occurred. It is possible that the gestures could have flagged events, however due to limited programming experience in C# it was decided to have these gestures being recognised continuously in one set function which could also change the Boolean value.

Building a 3D image is of course a vital part of this project, and various references were used that aided in building a 3D image. In particular the Microsoft website had a number of generic tutorials to help, and the website reference [25]. It was found that images could be built using libraries that could be purchased from third parties; however it was decided to instead try and build a program that used the Microsoft WPF format. This method involved creating a number of vertices that then create a mesh. This mesh can then be modified in colour and shape. A generic shape was chosen which can be modified if the vertices are also changed in the main MKIMP code. It is also quite difficult to build complex models, though this could be something that is worked on in the future.

Originally the image, once built, was displayed on the front panel using XAML functions. This in itself was not useful, but it was a starting point. The next stage was the ability to be able to manipulate these 3D images, and firstly this was done using mouse imitated events. This has been removed from the program now, but during the development stage it was the easiest way to develop the program. Using parts of other programs, it was possible to build methods that could track the mouse position and allow this to rotate the image and also track the mouse scroll wheel to allow the image to zoom in and out. It took approximately a week to have something that functioned as intended, and it was found that some of the code from other programs was in fact using poor structure and better formats were developed. This included using better syntax and methods or functions that could complete multiple tasks at once.

Finally the collation of the gestures and the 3D image took place, and this was by far one of the most intensive areas of development of this program. Firstly it was found that the methods used in the program currently were not ideal as they could have been more iterative. Also many methods of finding better ways to rotate and move the image were trialled until the current iteration was decided upon as listed in Table 2. Next it was found that the gestures did not necessarily equate to how fast the rotation would occur or how far it would move. Again trial and error was used to achieve a relatively smooth method of transitions. This included significant time in finding how to zoom in and out without the image moving too fast. One of the main issues is that the zoom was much too fast and the image would disappear before it could even be seen by the user.

The gestures were also finetuned and made more relevant once the 3D image manipulation was possible. This was done by thinking more about what was an intuitive or natural way for a user to manipulate a 3D image. These new gestures then needed to be finetuned again to be able to manipulate the image somewhat regularly, and to the user's desires. As mentioned overall the development of this program took a long time, and could have taken many more hours. It was decided in the closing week of the thesis project to concentrate on making the program more presentable to conduct a demonstration of the programs.

5.4 The Project Progress

The latest revision of the project can be seen in Figure 15. It shows that there are individual lights for each gesture that light up when initiated, and there is a feedback screen in the form of a 'skeletal' tracking image. There is also the Manipulated Image panel that allows the user to see the image that is being manipulated, and how it changes with their gestures. This completes all the basic objectives initially set for the 2nd program.

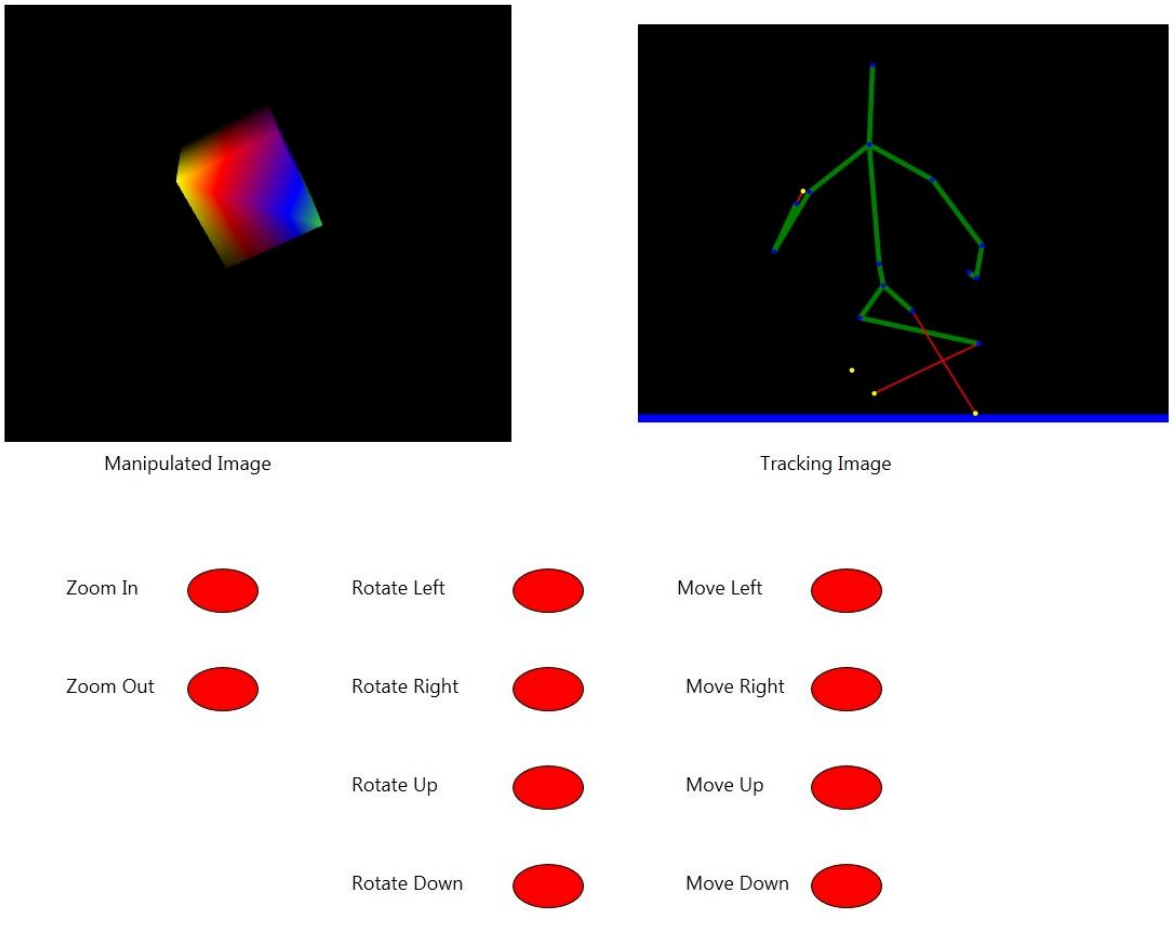

Figure 15: Kinect Image Manipulation Program Interaction Panel

Since this general form of panel has been developed, and the code well commented in the adjoining appendices, it is possible that the gestures can be easily modified by someone who can program, to whatever they desire. Extra features can be added to the program such as rotating about the Z-Axis, or initiation gestures for the general gestures. It is also possible that the manipulated image can be modified, rather than having a simple cubic function, although colour and lighting can be changed to give better definition of the object.

5.5 Program Improvements

Since the program has been developed to a level that is accessible and operational, it is appropriate to consider program improvements that could be made on this particular part of the project. Most of the improvements that could be made to this project are in fact additions; however both actual improvements and additions that are directly related to this current project shall be discussed.

With regard to improvements, the main area would be how the program is structured and organised. Learning C# and how to implement it was very difficult in itself, and therefore as a relative beginner of this language there are many structural peculiarities when comparing this program to others. In particular, improvements such as having intuitive gestures that can be recognised, and gestures developed as their own method or function would have been easier to utilise. By enabling this it would have allowed the gestures to be called more freely thus decongesting the main program. It would also enable an event change to be registered, thus creating a gesture recognition point only when a gesture occurs, as opposed to the method used in the program that sets a Boolean variable true if gesture recognised or otherwise false.

Other general structures of the coding may also run more efficiently, however that is outside the scope of both this report and the researcher's knowledge. With this ability to structure the code better the program would run more efficiently and be less likely to slow down. It could also be easier to add additional code to the program, as other methods could be used repeatedly in order to accelerate the program and programming. For example it could be possible to record gestures that a person makes and then enable them to be assigned to specific functions of the user's choosing. For example: a user could record a gesture of them moving their hand from the top of their head to their stomach, and this gesture could then be assigned to scrolling down the program page.

Better filters could be introduced that are able to filter out very small changes that are not meant to be recognised. This will aid in rejecting false positives, and prevent unwanted actions occurring or gestures be recognised. More time should be spent on changing not only the filter, but the gestures themselves so that they are more intuitive and less likely to lead to false positives in the first place. This would require many hours of study in the field

of biomechanics, behavioural biology and natural algorithms. This of course would be very difficult and could develop into a thesis within itself.

Finally the actual layout of the program could be improved. This includes the object that was used for image manipulation, as it should be possible to choose a colour and perhaps even an image or object of choice. This would not be difficult if available libraries were enabled that allowed easy input of 3D imagery into the IDE. The layout could also be improved to provide more aesthetical indicators and displays; however this requires a lot more time spent in becoming familiar with the Design section of Microsoft Visual Studio 2010. This is not the purpose of the project.

Overall there are several improvements that could be made, and will be further discussed in the Future Project Scopes section. The Future Project Scopes section details what can be built upon from this program and even includes completely new areas that can be researched, but still rely on the information obtained in this report.

Chapter 6: Outcomes of the Thesis

This chapter is designed to give an overall description of the final outcomes. This is not limited to the project itself; it also considers the personal outcomes from completing this ambitious project. Finally, this chapter discusses the difficulties that were found in both conducting and completing this project.

6.1 Overall Project Outcome

The programs had their issues in being developed, and there were many other hurdles along the way, some of which will be discussed in a later section. Overall the project was a success because it is able to give a user the general function of what it was first intended to do. The program is able to manipulate a 3D image using the Microsoft Kinect sensor, all through self developed programming skills.

This project was overly ambitious to begin with, and a lot of the objectives that were first discussed were instead moved to become future projects. As a project that is in a new area, and not attempted by any student at the Murdoch University before, it is difficult to understand what is truly required to create a successful project. This is not an issue, and it is hoped that from reading this thesis that a future student undertaking similar research will not make the same mistake of trying to accomplish too much at once.

Although the programs did not have as many functions as intended, the basic functions are present and that is important. In fact there were many other functions developed along the way, as was discussed about in the program development sections, which have since been removed in order to make the programs more efficient and simpler to be understood. This testing was essential along the way to truly understand how the program would function in real world scenarios.

Unlike other projects this project did not contain a lot of empirical data, and this was due to the fact that the method in which the project was developed required extensive trial and error. Empirical data was not necessarily useful, as it did not lead to completing any of the objectives; however the idea of testing the sensor for statistical accuracy is discussed in future projects area.

6.2 Personal Outcomes

A very important area of conducting any research is reflecting on what was learnt. This includes not only what was explicitly learnt from the research material, but also what was learnt about ourselves. Undertaking and completing a major project is an achievement in itself and it is hoped that on reflection that we can learn from our mistakes and weaknesses whilst conducting the research, but also find our strengths that we can develop further into the future.

From this project it was possible to identify a number of weaknesses, including:

- Ability to keep detailed records
- Having the self assurance that what was completed is sufficient
- Working at a consistent pace

These weaknesses were independently addressed during the project, but also shall need to maintained and improved in the future. With regards to keeping details, it was ensured that all data relating to the project was located in a set folder and also had a diary that was maintained almost daily. With regards to self assurance of completing enough tasks, set tasks that needed to complete each day were listed, and then assured myself that it was enough work. This is an area that will continually need to be improved, as having too high a set of standards can have a negative effect on morale. The most effective work produced tended to be in short bursts, and therefore realising this frequent short 5 minute to 10 minute breaks every hour to maintain a more consistent work pace and stopped work for the day when my set day tasks were completed. This will be a method of self monitoring that shall also be utilised in the future.

Also from this project it was possible to identify a number of strengths:

- Time Management
- Planning Schedules
- Pace of work
- Ability to learn new concepts very quickly

By recognising these strengths it is possible to utilise them as best as possible for circumstances such as this project. By having strong time management skills it enabled me to be able to complete a thesis and two units whilst working 15 hours per week and raising a young family in just one semester. By having stringent planning schedules, no assessment or task was late and was finished on or before the set date. By working at a fast pace, large amounts of work were able to be completed in short amounts of time. Being able to learn things very quickly shortened the amount of time required for certain parts of the project. Of course all these skills will be strengthened further into the future, so as to allow me to become a more efficient and effective learner and a more productive person.

From this thesis I also learnt many skills that I shall be able to develop in the future. Some of these will perhaps not be used past the scope of this thesis, but others may be required for further work or even find place as a pass time. These included:

- Being able to explain my project status to a supervisor
- Being able to briefly demonstrate my current project capabilities effectively
- Being able to develop effective Gantt Charts
- The ability to develop small projects in C# that are relatively robust
- The habit of documenting code and commenting individual lines or functions
- Being able to develop programs in Microsoft Visual Studios 2010

Overall this project taught me a lot about myself and how I am able to cope with numerous simultaneous tasks and stresses. It showed me that I have built a strong foundation for being a productive member of society in the future, by working both hard and smart. By managing my time effectively, I was still able to spend time with my family and enjoy my hobbies (squash!) without feeling any part of my life was neglected. I now know that I can work hard and produce results whilst still maintaining a balanced life that I can enjoy.

6.3 Project Difficulties

This project was quite ambitious in its original idea and intentions as can be seen in the original project scope and objectives and the subsequent project revisions. It was overly ambitious because a lot of tasks were required to be completed in the 15 week time frame. The enormity of the information required to comprehend the project scope was misunderstood, as was quantity of the work required to complete the objectives. This caused some of the original objectives to be left out of the final program, although the basis for these ideas is available for use in the next iteration of this project.

Although the information available was vast, a lot of the information was actually out of date. As mentioned earlier, the Kinect was released in 2010; however the Windows SDK was not available until June 2011 [17]. Furthermore the most current version and version used for this project, Windows Kinect SDK Version 1.5 were not available in Australia until May 2012 [17]. This meant that even though information had been created and tutorials developed, a lot of the information was found to refer to the June 2011 version. There was also information pre-dating this period, however these developments relied on 3rd parties that accessed the Kinect directly and were not able to easily connect to the Kinect sensor reliably. This initially caused a lot of confusion, as items that seemed to be straight forward were unable to be completed in the most recent version due to changes in the SDK.

To make matters worse, some of the libraries in the latest version were completely deleted or renamed. As a beginner, unfamiliar with the C# language and structure, this made recreating or even running sample programs quite difficult, as the libraries and classes needed to be modified so as to refer to the correct reference. This change in structure was not documented clearly, least of all on Microsoft's website, which led to a lot of wasted time. There were only small parts of information that did detail this change on some programming forums such as stackoverflow.com. As the latest kit was only released a few months before this project was started, there was a lot of information that was required to really get the project moving that simply didn't exist. It is unlikely that this information will be produced in the near future, and in fact may take another full year until more tutorials and help documents are created.

Of course the final difficulty in this project was time. The management of the time available was successful as the main objectives were completed, however the amount of time specific tasks took was longer than expected. This caused some parts of the project, such as sound imaging or speech recognition to be completely removed and are instead included as future proposals for projects. By still standing by the schedule, parts needed to be removed in order to achieve the final product of the Kinect Image Manipulation Program. This cannot be avoided as a project of this nature had not been undertaken by the student, in the future when attempting a project such as this; perhaps more realistic individual time frames can be developed.

Chapter 7: Concluding Remarks

7.1 Conclusion

The purpose of this thesis was to ultimately create a program that could interface with the Microsoft Kinect, and recognise gestures autonomously that would allow a user to manipulate a 3D image. The project required many areas of research and a steep learning curve in understanding how to program in C# and Microsoft Visual Studios. The overall project developed just from an idea, however having just an idea was not enough, it needed to be developed through thorough planning and self discipline.

The development of the Kinect Interface program was completed without issues, and allowed an insight into the how the Kinect actually functions. Overall the 3D image manipulation program functioned, however not precisely as intended. This is not an issue as the precision was not outlined in the project plan or progress report as an objective, although this is discussed in future project scopes. It was found that as technologically advanced as the sensor is, it is not capable of everything that can be conceived. For example the Kinect sensor is unable to operate as fast as a user can make a movement. Therefore through this understanding it is possible to better plan future projects.

The project involved many areas of learning, including the sensor itself, its programming interface, the syntax of the programming languages and self discovery. That makes the project an overall success, because it covered all areas that are required to be a useful research project.

It is hoped that this thesis will pave the way for future students to discover for themselves the powerful capabilities of the Microsoft Kinect coupled with the strengths of the C# programming language. It should be used as a reference to providing the student with initial knowledge, a set of well documented foundation programs and quick access to external references that will be required for the project. In the future the student should endeavour to use their imagination, innovation and creativity to design a project that challenges the device and themselves.

7.2 Future Projects

There are many possibilities with this project going forward, as it has only just began to delve into the full capabilities of the Microsoft Kinect and C#. Further projects are limited only by the next researcher's imagination as to what can be achieved, and this is supported by the fact that Microsoft is aiming to release the Kinect 2.0 with better resolution sensors and other improvements next year. In order for the next researcher to have some guidance, a few suggestions are discussed below:

Learning Device: The proposal for this idea is that an interface between the Kinect and a windows application is built so that the sensor data can be viewed by students studying ICSE degrees so that they can view the data without having advanced knowledge of C# or Microsoft Visual Studios. This interface will allow a user to select options such as what display they want to view, what the joint position is, the velocity of the joint or many other options. The data would not be able to be used directly by the students.

SCADA Interaction Device: This project could be aimed at enabling a user to control a SCADA operation through gesture-based control with the Kinect. This would allow operators who often control numerous processes through various computers, to control them through a single interface, alleviating stress. It could also keep the operators stay more alert, who often have trouble keeping focused due to the monotony of the position, by monitoring their eye movement and thus understanding their current mental state (circadian rhythm position).

Further Develop Medical Image Manipulation: It would be possible to continue the current project and develop it further. This could be done by incorporating the original theme, which is the ability to control 3D medical imagery. This could be done by using the current program and building upon it and changing the current general purpose 3D image for a medical image. This will be complex as the DICOM images that are used in the medical imaging world are typically made up of slices, such as in a MRI scan. Therefore it will be necessary to develop algorithms that will convert the slices and the position within the slices into simple easy to display set of vertices.

Development of more complex gestures: The gestures used in the 3D image manipulation program were quite basic. This was due to the limited time available, however as the preliminary research has been completed it would be possible for a student in the future to continue working on the gestures. The gestures could be more complex and more intuitive, with higher levels of accuracy. These gestures could be used in further programs that are developed either academically or commercially.

Accuracy testing of the Sensors: The reliability and accuracy of the Microsoft Kinect sensors is another possible area of research. As the sensors react at 30Hz this is typically fast enough to pick up most movement, though how accurate is the position of each joint? Is it possible to improve the recognition of gestures by understanding the reliability and accuracy of the actual sensor?

Transfer of data to other programs: Having to learn C# or another complex language before being able to utilise the data of the Microsoft Kinect increases the length of any research considerably. Therefore it would be ideal to create a project that enables the transfer of Kinect data to another program such as Matlab or Labview, programming languages that are taught as part of the ICSE or other electrical engineering curriculum. By creating this program, future students will easily be able to start manipulating the data so as to create interactive programs at a much faster rate due to not having to invest the time to learn a new programming language.

Voice Recognition Manipulation: As talked about earlier in the project, an original idea was to incorporate the built in voice recognition capabilities into a program that could also manipulate an image. An entire project could utilise the control systems already designed in this project, but change the method of manipulating the controls via voice commands. This would be a difficult project that would take dedication and commitment; however it is a project that has extensive possibilities.

Visual Device: The Microsoft Kinect sensors are not necessarily special in their own right, however combined in the way in which they are enables them to perform something unique at its current availability and price point. Utilising these unique features it would be possible to use the Microsoft Kinect as a Visual Device – or an aid device. This could be practical in robotic applications as the sensor is able to detect obstacles through depth perception. To aid in the depth detection at close range an aftermarket lens is available called the Nyko Zoom [29] adapter that causes the depth range to reduce enabling detection of closer items.

Bibliography

It should be noted that many references were made to forums throughout the project, in particular stackoverflow.com; however there is no specific page that was referenced.

[1] Webb, J., Ashley, J., *Beginning Kinect Programming With the Microsoft Kinect SDK,* Apress, 2012

[2] Bell, D., Parr, M., *C# for Students,* Addison-Wesley, 2009

[3] Microsoft Research, *Programming with the Kinect for Windows SDK*, Microsoft, 2011 http://research.microsoft.com/en-us/events/fs2011/jancke_kinect_programming.pdf

[4]Channel 9, *Kinect for Windows SDK Beta 2 Quick starts,* Microsoft, 2011 http://channel9.msdn.com/series/KinectSDKQuickstarts

[5] CodePlex, Coding4Fun Kinect Toolkit, Microsoft, 2012 http://c4fkinect.codeplex.com/

[6] Khoshelham, K., *Accuracy Analysis of Kinect Depth Data*, University of Twente, 2011 http://www.isprs.org/proceedings/XXXVIII/5-W12/Papers/ls2011_submission_40.pdf

[7] Microsoft, Microsoft Kinect SDK Documentation, 2012 http://msdn.microsoft.com/en-us/library/hh855347.aspx

[8] C# Station, C# Station Tutorial, 2012 *Data Structures and Algorithms Using C#,* http://www.csharp-station.com/Tutorial.aspx

[9] Openkinect.org, Kinect Protocol Documentation, 2011 http://openkinect.org/wiki/Protocol_Documentation

[10] Benko, H., *High-Fidelity Augmented Reality Interactions,* Microsoft Research, 2011 http://research.microsoft.com/en-us/um/redmond/events/latamfacsum2011/presentations/day_2/barahona_3/hrvoje_benko.pdf

[11] Tsikkos, M., Glading, J., *Writing a gesture service with the Kinect for Windows SDK,* MCS UK Solution Development Team, 2011 http://blogs.msdn.com/b/mcsuksoldev/archive/2011/08/08/writing-a-gesture-service-with-the-kinect-for-windows-sdk.aspx

[12] Prime Sense, Accessed October 2012 http://www.primesense.com/

[13] McMillan, M., *Data Structures and Algorithms Using C#,* Cambridge University Press, New York, 2007

[14] Date to Date Calculator, timeanddate.com, Accessed October 2012

http://www.timeanddate.com/date/durationresult.html?d1=30&m1=7&y1=2012&d2=16&m2=11&y2=2012

[15] Wikipedia, *Microsoft Visual Studio,* Accessed October 2012

http://en.wikipedia.org/wiki/Microsoft_Visual_Studio

[16] Visual Studio Resources, Microsoft, Accessed October 2012

http://msdn.microsoft.com/en-us/vstudio/cc136611.aspx

[17] Wikipedia, *Microsoft Visual Studio,* Accessed October 2012

http://en.wikipedia.org/wiki/Kinect

[18] Microsoft Kinect Teardown, ifixit, Accessed October 2012

http://www.ifixit.com/Teardown/Microsoft+Kinect+Teardown/4066/1

[19] Davison, A., *The Tilt Motor, LED, and Accelerometer,* Davison, 2011 http://fivedots.coe.psu.ac.th/~ad/jg/nui16/motorControl.pdf

[20] Davison, A., *Using the Kinect's Microphone Array,* Davison, 2011

http://fivedots.coe.psu.ac.th/~ad/kinect/ch15/kinectMike.pdf

[21] Audio Stream, Microsoft, Accessed October 2012
http://msdn.microsoft.com/en-us/library/jj131026.aspx

[22] Davison, A., *Kinect Imaging,* Davison, 2011
http://fivedots.coe.psu.ac.th/~ad/jg/nui13/KinectImaging.pdf

[23] Konolige, K., Mihelich, P., *Technical description of Kinect Calibration,* ROS.org, Accessed October 2012
http://www.ros.org/wiki/kinect_calibration/technical

[24] McGrath, D., *Kinect's BOM roughly $56, teardown finds,* EE Times, Accessed October 2011
http://www.eetimes.com/electronics-news/4210649/Kinect-s-BOM-roughly--56--teardown-finds-

[25] Kindohm.com, *Windows Presentation Foundation 3D Tutorial,* Accessed October 2012

http://www.kindohm.com/technical/WPF3DTutorial.htm

[26] Microsoft, Microsoft Kinect, 2010

http://www.microsoft.com/en-us/kinectforwindows/

[27] Microsoft, Microsoft Visual Studios 2010, 2010

http://www.microsoft.com/visualstudio/eng/visual-studio-update

[28] Microsoft, C#, 2000

http://msdn.microsoft.com/en-us/vstudio/hh341490.aspx

[29] Nyko, Nyko Zoom, 2011

http://www.nyko.com/products/product-detail/?name=Zoom

[30] Stroustrup, B., C++, 1985

http://www.stroustrup.com/

[31] Wikipedia, *C# Programming Language*, Accessed December 2012

http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29

[32] Radeck, K., *C# and Java: Comparing Programming Languages,* Microsoft, 2003, Accessed December 2012

http://msdn.microsoft.com/en-us/library/ms836794.aspx

Appendices

Please refer to the attached CD for the following appendices:

A1 – Code for Microsoft Kinect Interface Program

A2 – Code for Microsoft Kinect Image Manipulation Program

A3 – DICOM MRI Brain Image