Engineering Thesis, 23/12/2014

To Build a Gaming Human Interface Device, Designed for those with Limited Dexterity

A report submitted to the school of Engineering and Information Technology, Murdoch University in partial fulfilment of the requirements for a Bachelor of Engineering Degree.

Author: Daniel Van't Sant
Supervisor: Gareth Lee

Ethics Permit Number: 2014/181
RAMP Reference: RAMP 0123

Abstract

With the competition in the games market focus is placed on entertaining the user for the sale of their products. This brings entertainment to the safe environment of the home. To play these games the user needs to use a keyboard and mouse.

For somebody without finger dexterity and limited arm movement, pressing keys on the keyboard is impossible without pressing other keys at the same time. The mouse is also impossible to use.

This project involves the design of a wireless controller made for an individual who has no finger dexterity, limited arm movement and has access to an optical head tracker to move the mouse cursor. The inputs are done by buttons and joysticks. The project requires careful design and testing to ensure the inputs can be operated by this individual.

The average time spent comfortably playing on the game controller was four hours. The delay coming from the controller was not noticeable for all the button presses. The processing time to do a key press by the controller was tested to be less than thirty two milliseconds. The battery that is in the controller will last for 16 hours of gameplay. The controller is mostly operated by the user so that less work is done by the carers. The only other method of playing games before the controller was to have a pointer glued into his splint. This could then be used to operate the mobile phone by using the pointer on the touch screen. The user found that computer gaming is more entertaining than playing games on a mobile phone.

Contents

Abstract 1

i) Declaration 3

ii) Acknowledgements 3

List of Figures 4

1. Introduction 6

  1.1. Available Products 7

  1.2. The Joy of Video Games 8

  1.3. Foundations and Charities 8

  1.4. Ethics 9

  1.5. Project Design 10

2. Selection of Components 12

  2.1. Previous Work 12

  2.2. Arduino Micro 12

  2.3. Xbee – Wireless Communication 14

3. Construction of Hardware 15

  3.1. Left Controller 15

    3.1.1. Modifications 21

    3.1.2. On the Chair 23

    3.1.3. Problems Encountered 23

  3.2. Right Controller 24

    3.2.1. On the Chair 25

  3.3. Display 25

4. Program Development Steps 27

  4.1. The LCD 27

  4.2. Xbee Setup and Configuration 27

  4.3. Reading Inputs 30

  4.4. Using Millis() 31

  4.5. Documentation 32

  4.6. Grid of Nine Keys 32

  4.7. Xbee Sleep 35

  4.8. Profiles 38

  4.9. Menu 38

  4.10. Battery Voltage 40

5. Wiring Diagrams 43
6. Test Results 46
7. Input Lag 48
7.1. Assumptions 48
7.2. Results 48
7.3. Method 49
8. Problems Encountered 50
8.1. Xbee Connectivity 50
8.2. Ctrl, Alt and Delete Key Press 51
8.3. Batteries 51
8.4. Corner Keys 51
9. Future Improvements 52
9.1. Xbee Parameters 52
9.2. Customisation of the Profiles 52
9.3. The Left Enclosure 52
10. Conclusion 53
11. References 54

i) Declaration

I declare that the following report is my own work, unless it has been referenced.

ii) Acknowledgements

I'd like to acknowledge my brother for providing the funds required to buy the electronic equipment and enclosures used for the project. I'd also like to thank him for coming up with some ideas on how to make the controller compatible and for his decisions on the best possible placements of the buttons and joysticks.

This document is the product of my knowledge in electrical engineering so far. The expected completion date will be the middle of 2015. I'd like to thank my parents Vicki, Hank, Amanda and Robin for their support throughout my degree.

I'd also like to thank my supervisor, Gareth Lee for his help, experience and foreseeing the project as an engineering solution.

Arduino and Adafruit can be acknowledged together for bringing open source technology freely available for all purchasers with well documented examples and tutorial for their products sold.

List of Figures

Figure 1, Component Block Diagram 10
Figure 2, Packaging and Size Comparison with an Australian 20c Coin 13
Figure 3, Front of Arduino Micro 13
Figure 4, Back of Arduino Micro 13
Figure 5, Left and Right Controller Fitted in the Chair with Velcro 15
Figure 6, Old Design of Left Controller 16
Figure 7, Top View of the Controller After Un-folding Each Piece 16
Figure 8, Left Controller Made from Craft Paper 16
Figure 9, Left Controller Design using Google Sketch-Up 17
Figure 10, Template and Acrylic Piece to be cut 17
Figure 11, Acrylic Piece Marked 17
Figure 12, Drawing to find Angle between Faces 18
Figure 13, Band Saw used to cut Acrylic 18
Figure 14, The Angle of the Cutting Table can be adjusted 18
Figure 15, Final Piece Next to the Prototype 18
Figure 16, Showing that the Hot Glue didn't Look Good 19
Figure 17, Silicone Used on Left Controller 19
Figure 18, Ready to Apply Acri-bond 20
Figure 19, Design of the Base 20
Figure 20, The Base showing some Components Inside 20
Figure 21, Joystick and Base Mounted 20
Figure 22, Left Controller with Buttons Assigned 21
Figure 23, Different Joystick and Velcro Added 22
Figure 24, The Battery Voltage Divider 22
Figure 25, Neat Wiring Job 22
Figure 26, Left Controller in Arm Rest 23
Figure 27, Left Controller Locked in Arm Rest by Grooves 23
Figure 28, Back of Left Controller Locked onto the Chair's Joystick Base 23
Figure 29, Joystick is reachable by the User (Showing the Researcher's Hand) 23
Figure 30, Drilling the Base Broke the Piece 24
Figure 31, Right Joystick - Penny and Giles, JC2000 24
Figure 32, Right Joystick with Button 14 25
Figure 33, Inside Right Joystick 25
Figure 34, Right Controller fitted to Arm Rest 25
Figure 35, Right Joysticks Handle Locks the Users Hand (Researcher's Hand) 25
Figure 36, Inside the LCD Display 26
Figure 37, First Menu Page 26
Figure 38, Second Menu Page 26
Figure 39, Third Menu Page 26
Figure 40, Minecraft Profile showing the Grid of Nine Key 26
Figure 41, FTDI to USB Cable 28
Figure 42, Xbee Adapter from Adafruit 28

Figure 43, Xbee Adapter Soldered onto Headers 29
Figure 44, Testing the Components Together on the Breadboard 30
Figure 45, Using millis() to Flash a Heading 31
Figure 46, Using millis() to move down Menu 32
Figure 47, Joystick Position to Grid of Nine Keys 32
Figure 48, Code for the Negative Trigger 34
Figure 49, How `lockaction` is set to Prevent the return to Centre Grid 35
Figure 50, Serial Monitor to Xbee to Program Sleep Mode 35
Figure 51, Commands to turn on Sleep Mode 36
Figure 52, Sleep after One Second and Write to Xbee 37
Figure 53, Code to read Joystick, Buttons and Voltage from Xbee 37
Figure 54, Menu flow Diagram 39
Figure 55, Diagram of how the Menu Initialises 39
Figure 56, Scaling Analogue to Digital Battery Voltage 41
Figure 57, Log of Battery Voltage versus Hours of Operation 42
Figure 58, Display's Wiring Diagram 44
Figure 59, The Controller's Wiring Diagram 45
Figure 60, Input Delay from Button presses 48
Figure 61, Commands to Restore the Xbee and Setup different Addresses and Channels 50

1. Introduction

This project is concerned with building a computer game controller with joysticks and buttons that is compatible with a user that has quadriplegia (Apparelyzed, n.d.). The controller is to be wireless from the computer making the setup easier and the wheelchair more mobile.

Commercially available products were considered before this project was started. The controllers researched were badly designed and were not worth buying. The realisation to build a new controller with a new design was considered to be the best option. This way any problems can be foreseen and be solved. This required lots of meetings between the researcher and the user. The computer game controller is designed for an individual who has quadriplegia (Apparelyzed, n.d.). The controller may not work as well for somebody else with quadriplegia because the symptoms may be different for another person.

For this project the user has the ability to move his arms (not above the chest) but not move his wrists. He has access to a SmartNav (2012) which is an optical tracker that tracks the user's head movement and in turn this moves the mouse cursor. The intended user can only do a single operation with one hand because his fingers cannot move. This constrains the controller design but does not make gameplay impossible. Conventional games need three main controls to manipulate the game character, these control:

- where the player is looking
- the player's position
- the player's action

With the SmartNav controlling the direction the player looks in the game, the two spare hands can do the other operations. For this project the person's movement will be done with the right hand using the joystick and the actions done by the left hand. These three needs were carefully thought through thus providing a solid foundation in the design and implementation of a successful overall control interface.

One of the functions of this controller is that nine different key presses can be activated by a button, joystick and a LCD screen. With this design the controller can be made small and lots of buttons can be pressed for different actions in the game. If instead nine big buttons were used, the enclosure required would be large requiring a longer reach from the arms. Using the joystick to have nine selectable key presses means less arm movement is required and the controller can fit on the wheelchair's armrests.

Some games require different key presses. The plan is to have a single game profile working, then adding a menu system for the player to choose different game profiles. The game profiles will map different key presses for each button. This was completed and could be controlled by the user. This made the controller usable for different games.

This project calls for the selection of electronic devices for their purpose, research in connecting devices together for communication, programming, manually soldering components together and soldering wires to the correct terminals. Documentation was kept on day to day activities as well as the wiring diagrams. The code was split into segments with a heading to explain what each part of the program does. This was useful for debugging the program.

1.1. Available Products

Some quadriplegics cannot move their arms. The Quadstick can be controlled for people without arm movement (Davison, n.d.). The Quadstick is designed for a quadriplegic that has no arm and leg movement. The user plays games using sip and puff sensors and a chin joystick. This device has been developed by Ken Yankelevitz for 30 years. His work has been passed onto Fred Davison due to Ken Yankelevitz' health conditions (Quad Control, n.d.). Mark Barlet from Stephen Dockery's article (Dockery, 2011) explains how it is one of a kind and how Ken Yankelevitz solders and pieces together each controller. He has to do this because the gaming manufacturers are not interested in developing accessible controllers because of the small market available. Ken Yankelevitz has built over 800 devices in his time (Dockery, 2011). A picture of the Quadstick can be found on the website by Fred Davison (Davison, n.d.). Like the Quadstick air controlled switches could be used for this project. The reason this thesis will not use air controlled switches is because the target user has some arm mobility to exploit and does not like the idea of spit building up in the tubes.

Some other game controllers are available on the market can be seen in the website from Broadened Horizons (Broadened Horizons, n.d.). These controllers look as though they are designed for somebody with good arm movement since the buttons are close together and close to the joystick. The buttons in the corner are lower than the rest. These buttons would be hard to press without pressing the upper buttons. This controller would be difficult to use, for the person for which this project is designed for.

1.2. The Joy of Video Games

The book by James Paul Gee (Gee, 2005) describes that gaming is the melding between the character created in the game and the person playing the game. The game developer creates a domain for the game character while the player can then choose which identity he will play. The book goes on to talk about the set of rules and game's history of events that overlays the relationship between the game's character and the person playing. This creates a fictional career in the game. Some games have a certain discipline to follow, this creates a fictional professionalism between the player and the game character. One example given was the doctrine of the U.S Army. This evokes a strategy for the player.

Although games have different levels of focus on strategy and identity it would be easy to draw the conclusion that for somebody who cannot often get out of the house, they would like to build a fictional identity which could be mixed with fun and humour in the ever growing social and networked gaming world we are experiencing today. With games now including push to talk and texting features it is becoming easier to find the social side of playing games.

When there are tasks to work together there is a sense of belonging. For example playing a network game of Minecraft, your friend may be stuck down a mine from which they are unable to escape. The other player realises this and crafts (builds) some ladders out of wood. The player then comes back and throws the ladders down to the player who receives the ladders and places them on the mines wall so he can leave the mine safely. A simple task like this would not happen in real life for a disabled person and it is helping each other develop, which is quite primitive in us all.

In his book James Paul Gee (Gee, 2005) introduces Tetris and explains why it was so successful. We all are trying to find patterns in our life that make sense. These life patterns are complex to grasp and most of the time cannot be put together. When playing a game like Tetris we get back the control in our lives for the time we play and feel a sense of achievement which is hard to get from our complex lives. It is these aspects of the game which can be good for someone, to play games for the identity, strategy and the social part of the game which maybe may be missing from someone's life.

1.3. Foundations and Charities

The ESA has statistics to show that 59% of Americans play video games (Entertainment Software Association, 2014). This shows that the population of gamers is large. Of this population there is a small group of disabled people which would also like to play games. There exists a charity that wants to help disabled users experience the same enjoyment and improve their quality of life. AbleGamers is a foundation whose purpose is to bridge the gap between disabled gamers and game developers (The AbleGamers Foundation, 2014). The president of AbleGamers, Mark Barlet has stated that quadriplegics make up a small portion of gamers that are pushing for accessible games and controllers (Dockery, 2011).

1.4. Ethics

Research projects at Murdoch University that involve testing with humans that may potentially cause harm need ethics approval (Murdoch University, n.d.). This gaming controller may cause muscle soreness for the user from reaching for buttons and joysticks. Hence ethics approval needs to be completed to test the controller. Testing will show if the controller can be played for a long period of time and to validate the controller’s function and design for playing games.

Before somebody can sign on behalf of a disabled person the Human Research Ethics Committee are concerned to know if the disabled person can comprehend the information given to them from the information sheet. The participant in this case has normal cognitive behaviour and will fully understand the information provided to him.

Another problem considered is that the family relation causes dependency issues which could lead to family problems if the user wishes to withdrawal from the test. To solve these dependency issues a third party participant was nominated in the application letter to sign or withdrawal, on the participants behalf. Since the tester is a close relation there may be pressure on him to test the device and give positive feedback for the test. From an engineering students’ point of view, negative feedback is encouraged to find the disadvantages of the controller so they can be fixed with future works. Therefore coercion did not exist since both positive and negative feedback is a benefit to the project. If he wished to withdrawal from the test, then the test would have been carried out by the researcher in order to finish the thesis. In this way there was no pressure on the participant.

1.5. Project Design

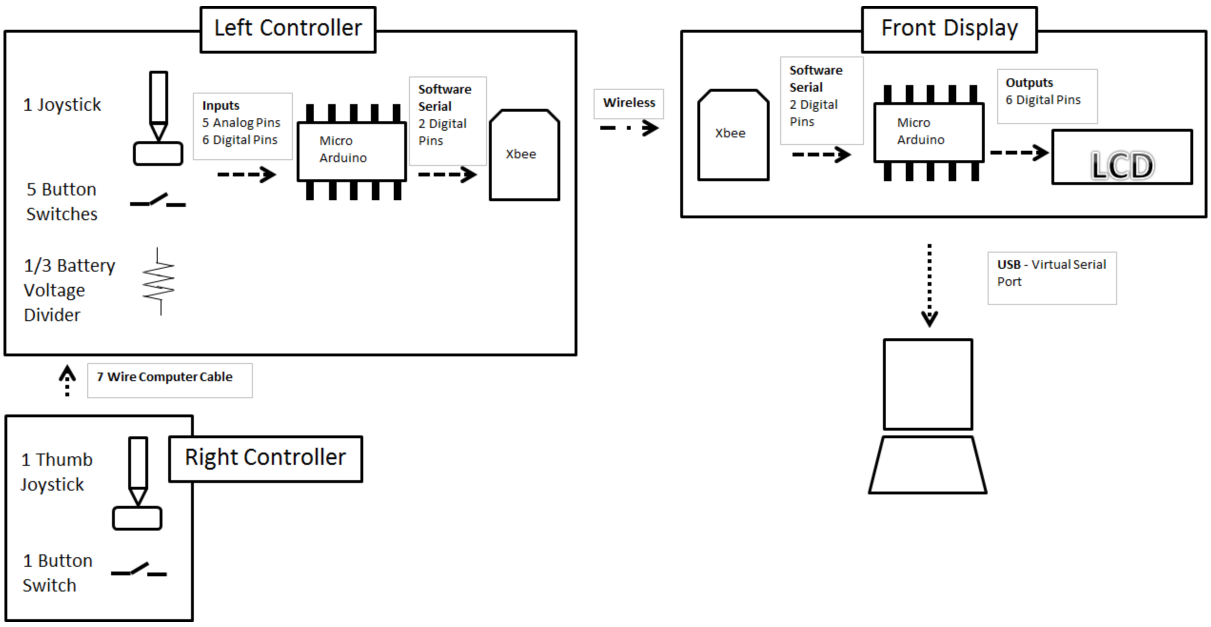

Figure 1, Component Block Diagram

Figure 1 shows the flow of data and the devices used within each module. There are three modules to be setup. The front display will have a twenty by four character LCD (Adafruit, n.d.) and be connected by USB cable to the computer. This will communicate wirelessly to the left controller which will house a thumb joystick and five buttons. The left and right controllers will be connected together by a 7 wire computer cable. The right controller will have a joystick and one button. Communication between the left controller and the front display will be done via two Xbee modules (SparkFun, n.d.). These will be connected to an Arduino micro (Arduino, n.d.). The Arduino micro in the left controller will have the task of reading all the inputs and then communicating this data to the front display. The front display's micro Arduino will hold game profile information, display some data and control the computer by using the Arduino micro's keyboard and mouse libraries (Arduino, n.d.).

A function of the controller is to type the W, A, S, D keys used to move the character up, left, down and right respectively. The user will be able to type these keys by moving the right joystick in the desired direction. The right controller will also have a button to press the escape key to leave the game. The buttons are manufactured by Ablenet and are easy to activate by pressing down anywhere on the face of the button, they are designed for people with low dexterity (Ablenet, n.d.).

The left controller will have five buttons and a thumb joystick. The thumb joystick was later changed to a stick joystick. One button will be for 'action' and the other four buttons are for a single key presses assigned by the game profile which is selected.

The LCD will display a 3 by 3 grid of keys that can be selected. The centre key will have a cursor next to it by default. To move to the other keys around the centre of the 3 by 3 grid of keys, the user will move the left joystick to the direction of the key in the display then let the stick move back to centre. The outer key that the joystick has moved to will now have the cursor next to it, after 10 seconds the centre key will have the cursor again. Another way the centre key can be selected is by releasing the action button. This means by pressing the action button when the cursor is next to the outer key, that outer key will be pressed until a release of the action button. Once the action button is released the cursor will move next to the centre key. This means the centre key can be pressed by the action button straight after pressing an outer key by releasing and pressing the action button again. For efficiency the most common key will be placed in the centre of the 3 x 3 grid. The centre key can simulate a key press by pressing the action button if the joystick has not moved in the last ten seconds or the user has released the action button (returning the cursor to the centre).

While observing the controller being used a common key press such as 'jump' or 'melee' worked well as the centre key because it can be pressed quickly without having to consume time moving the joystick.

Nine action buttons may seem like too many, but in many games this is how many keys are needed. For example the list below shows some generic key layouts:

1. Reload (R)
2. Grenade (G)
3. Action (E)
4. Select Weopon 1 (1)
5. Select Weopon 2 (2)
6. Investigate (Q)
7. Melee (F)
8. Jump (Space Bar)
9. Special Menu (Tab)

2. Selection of Components

This section describes a previous project to achieve a single joystick to keyboard controller for a previous engineering unit. This was the first attempt to create a joystick which could be used by a disabled person to simulate W, A, S and D keyboard presses. The work was not completely successful but the problems were revised when selecting components for this project.

2.1. Previous Work

An initial preliminary attempt at this project was done in the engineering unit "ENG306: Real Time and Embedded Systems". The project consisted of a microcontroller (68HC12) programmed using the language 'Forth'. This language was written in the SwiftForth IDE (Integrated Development Environment) (Forth Inc., n.d.). The program was written to read the joystick position and output this data as serial information. The serial data was transmitted through a Dual Asynchronous Communication Interface Adaptor (DACIA) to an RS-232 Port on the computer. Both of these devices were manufactured by New Micros (no longer in business), the models numbers are:

• Microcontroller :	NMIS-L-0912
• DACIA :	NMIS-5002

The RS232 port on the computer will continually read either a 'W', 'A', 'S' or 'D', or for the corners of the joystick 'WA', 'WD', 'SA', 'SD'. If nothing was read this implies the joystick has not moved. The serial read and write example from LabView is a program development from National Instruments was then used to read from the COM port of the RS232 card in the computer. This information was received in the LabView environment via a number of case statements which would perform a key press based on the position of the joystick. The example code for the keyboard presses is available from the National Instruments website and this was manipulated accordingly (National Instruments, 2006).

The limitation with the project was that a key press then release was done every 20 ms while the joystick was held in one position. The game's character then does not move constantly giving a jumpy response to his movement. The press and release caused a jumpy response in most games. Another problem is that key presses from the LabView environment did not work in most games. The project was just completed in time, so relatively little research went into how to solve the problem. It was not a complete solution but much was learnt about communication devices and configuring hardware using the command and status registers. This experience was employed when developing code for reading and writing to the Xbee, which was used in the current project.

2.2. Arduino Micro

This project was originally going to have very similar functionality as described in section 2.1 with the components being cheaper and more readily available. The first year engineering unit "ENG109: Engineering Computer Systems" introduced students to the world of microcontrollers. The project was to program a clock with hours, minutes and seconds to display on the LCD which was coupled to the Arduino Uno. The usability in having open source information readily available on the Internet is what motivated the use of Arduino products in this project.

The preliminary attempt to this project was revised with the supervisor and the problems occurring could be solved by using an Arduino Leonardo. The Leonardo can create key presses through the

USB port by using a virtual serial port this is known as a Connected Device Class (CDC) driver (Arduino, n.d.). The Arduino Uno which is the commonly used Arduino board has a second microcontroller on-board to do the communications through USB. Hence the Arduino Uno cannot setup the device as a virtual Human Interface Device because it always behaves as a serial port. The keyboard and mouse libraries available for the Leonardo made this controller a good selection. This meant LabView was not required.

After discovering the Leonardo, the range of micro controllers from Arduino was researched until a Arduino Micro was found which is similar to the Leonardo but in a smaller form. This was selected to be an even better solution given its more compact design. As the system was to be wireless two of these were ordered with one to be used for the computer interface and the other to read the inputs from the controller. Using the same type of Arduino in the controller and display means only one spare is required if damage should occur to either microcontroller. For an understanding of the size, figure 2 shows the Arduino micro in its packaging next to an Australian 20 Cent piece. Also figure 3 and figure 4 view the Arduino micro from the front and back.

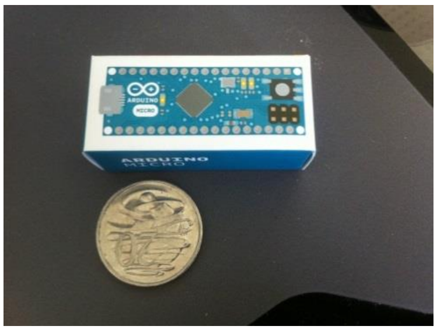
Figure 2, Packaging and Size Comparison with an Australian 20c Coin

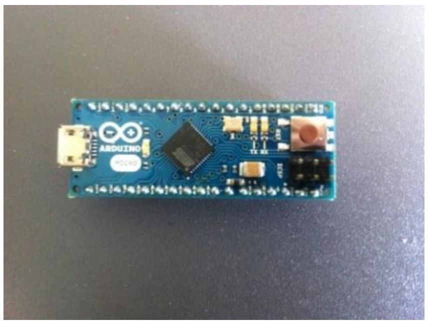
Figure 3, Front of Arduino Micro

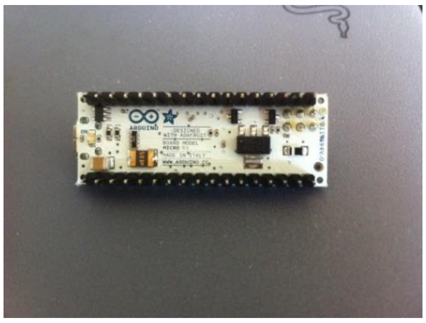
Figure 4, Back of Arduino Micro

2.3. Xbee – Wireless Communication

Making the controller wireless was desirable and offered a benefit to the user (see section 6). The Xbee wireless shields for the Arduino Uno and Leonardo were also discovered in the catalogue when browsing different Arduino boards to use. Curiosity in these devices lead to what the Xbee is and what can it do? As a result more research was done on the Xbee and it was found that they can do serial communication using two pins, transmit and receive. This was found in the video tutorials on how to setup the Xbee which consist of five tutorials (tunnelsup, 2015).

This project requires point to point communication where there is only communication between two devices. Mesh networking allows more than two devices on the network allowing communications between each device (Digi, n.d.). While researching the Xbee it became apparent that there were two different series, two different ranges and an option for the antenna (SparkFun, n.d.). There is a series 1 which is advised for point to point communication which was ordered for this project. The other is a series 2 which has more parameters to setup and can do mesh networks. The Xbee also comes in a Pro version with a 1.6 km range which uses more power (Digi, n.d.). The regular Xbee's range is 90 metres. The controller will be used within a couple of meters of the computer and will be networked between two devices therefore the series 1 was selected. The antenna selected was a trace antenna for both Xbees which is integrated in the circuit board. The other type has a short aerial which would make it hard to mount in the enclosure.

3. Construction of Hardware

Figure 5 shows a top view of the chair the controller is mounted on. This shows the position of the left and right controller in relation to the user's seating arrangement. To find the best position for the controller much of the effort went into observing the best location so that the buttons and joysticks are reachable.

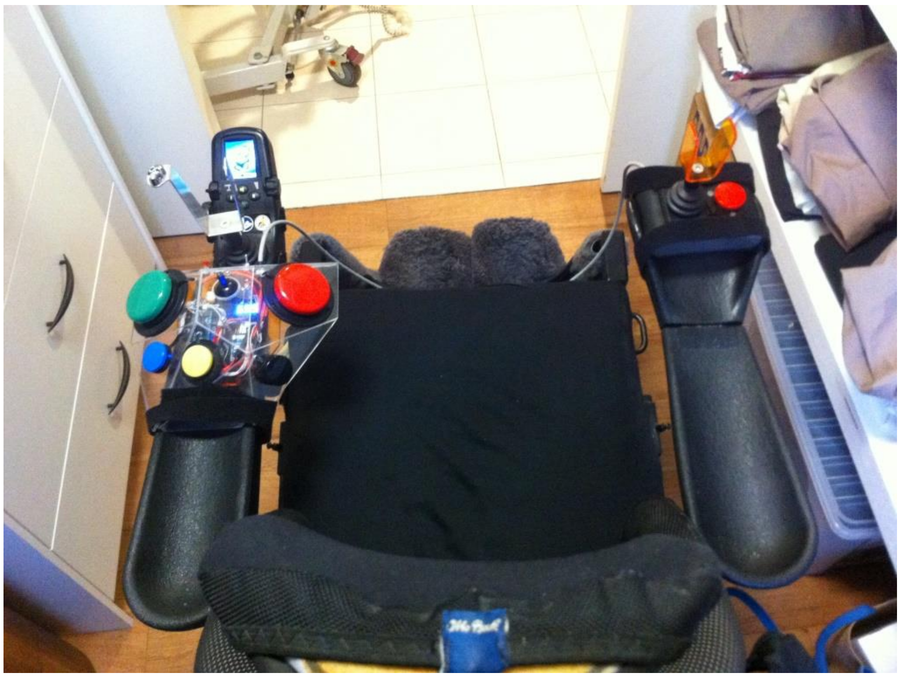

Figure 5, Left and Right Controller Fitted in the Chair with Velcro

3.1. Left Controller

The left controller was designed using a freeware three-dimensional drawing program called Sketch-Up Make (Trimble, n.d.).

To start the design a measurement was needed for how wide the base of the controller can be so that it will still fit on the chair. The width of the controller could be 23cm and the length 20 cm. The controller needs a joystick so a pyramid was thought to be the best starting point with a flat on top so the joystick can be operated. The height of the joystick was kept low so the left arm does not have to strain to use the joystick, but internally high enough to leave enough room for electronics.

Once a pyramid was formed the bottom left corner at the front was brought to a point and the three buttons were going to be mounted on the three surfaces of the bottom left corner. The point was brought to this corner since the user's arm is naturally pointing in when in normal rest position. Another flat was made as a consequence result. This flat was going to be used for an additional button. The old design with all the buttons mounted is shown in figure 6.

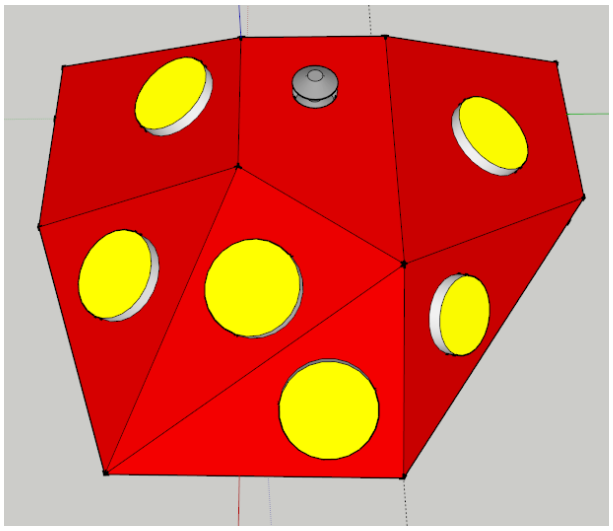

Figure 6, Old Design of Left Controller

A prototype was built by using a plugin for Google sketch-up called the 'unfold tool' this flattened out all the pieces, then a 1:1 scale print was done (Plugin: Unfold Tool, 2007). Figure 7 illustrates the template which was printed. Using the 1:1 scale print, the sides were cut out and this was used as a template against some crafting paper. By slicing the edges away and hot gluing the joins a prototype was made. Figure 8 shows the prototype made out of craft paper. This was used to find the best location for each button on the chair and to find the best place to mount the controller.

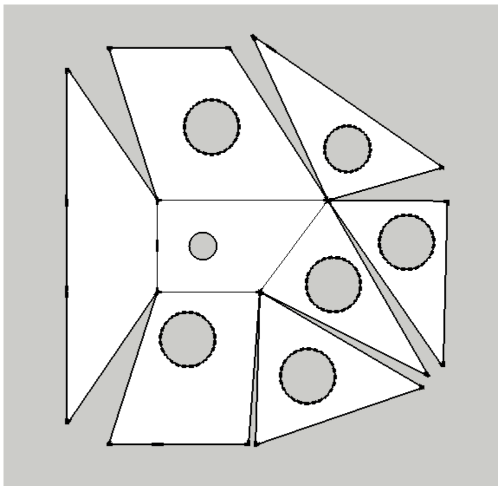

Figure 7, Top View of the Controller After Un-folding Each Piece

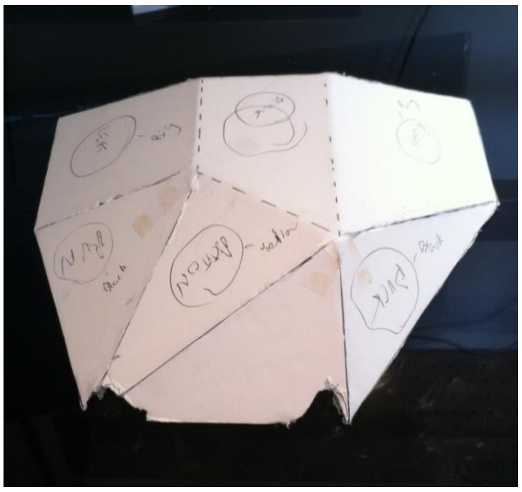

Figure 8, Left Controller Made from Craft Paper

Once the prototype was made and fitted to the chair it was realised that the front button would be too hard to reach and also allow a place for the user to rest their hand. This front turned out to be a good spot to strap the controller to the chair. The model was then updated and larger buttons were used for the mouse clicks as shown in figure 9.

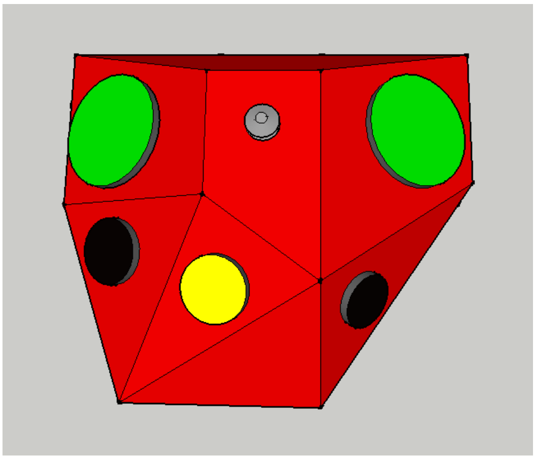

Figure 9, Left Controller Design using Google Sketch-Up

It was decided to make the controller out of clear acrylic to show all the connected electronics on the inside. Figure 10 shows the template which was used to mark the acrylic sheet. The marked sheet is shown in figure 11.

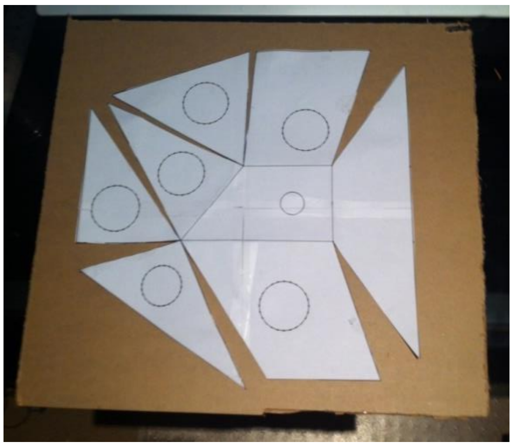

Figure 10, Template and Acrylic Piece to be cut

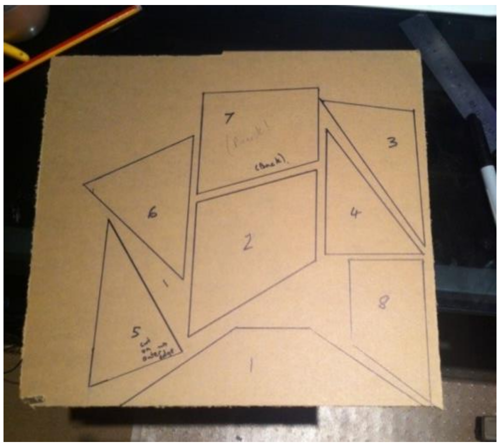

Figure 11, Acrylic Piece Marked

To stick the acrylic pieces together the glue was called Acri-bond (ATA, n.d.). This bond was used for the best visual appearance of the acrylic piece. The downside is that the pieces to be stuck need to be contacting each other. A really small gap will stick but machine lines need to be sanded. For this reason lots of effort went into making sure each edge of the acrylic piece was cut at an angle that made the surface of the join contact together. Sketch-Up Make was used to find the angle between each surface by pulling each surface up three dimensionally and measuring the angle between each piece using the plugin: Angle between Faces (Trimble, n.d.). Figure 12 shows the left controller design to find the angle between faces. Figure 13 shows the Band Saw used with a swivelling table to cut the pieces at a set angle. Figure 14 shows the protractor used to read the angle of the table.

Once the pieces were cut sticky tape was used to show how well they fit together shown in figure 15. At this stage one of the pieces needed sanding down further and all the other pieces needed to be sanded to remove machine lines.

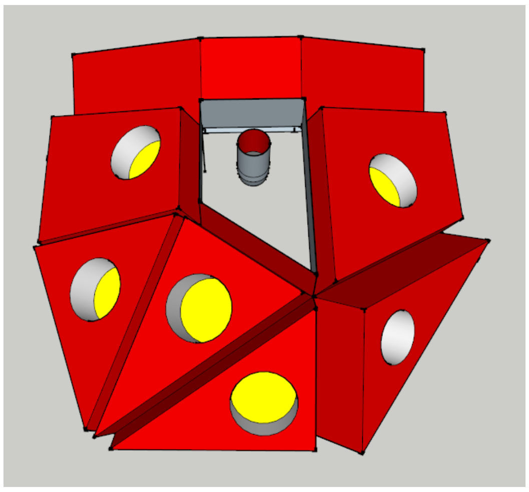
Figure 12, Drawing to find Angle between Faces

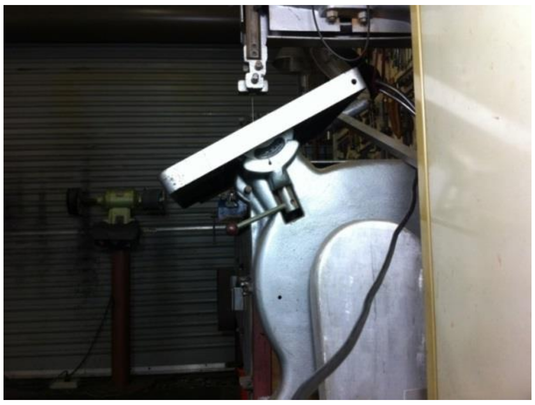
Figure 13, Band Saw used to cut Acrylic

Figure 14, The Angle of the Cutting Table can be adjusted

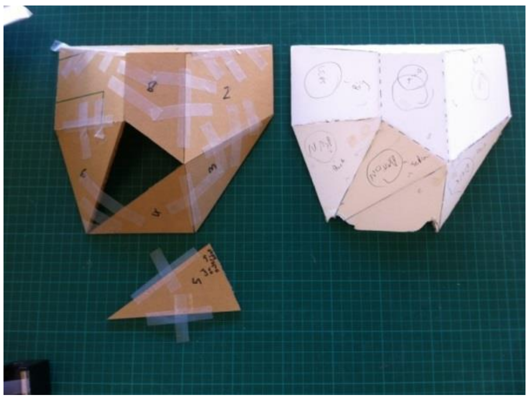
Figure 15, Final Piece Next to the Prototype

The Acri-bond was not used immediately because it was difficult to source the product. Instead hot glue was trialled. Figure 16 shows the poor results. After the hot glue was removed, silicone was instead used since it would be clear and hold the pieces together for the Acri-bond to set. Figure 17 shows the better results.

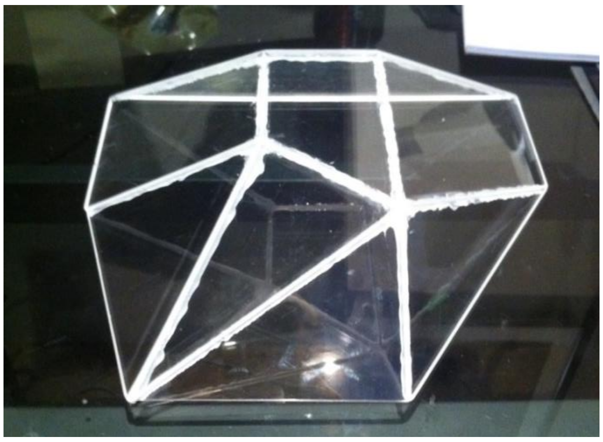
Figure 16, Showing that the Hot Glue didn't Look Good

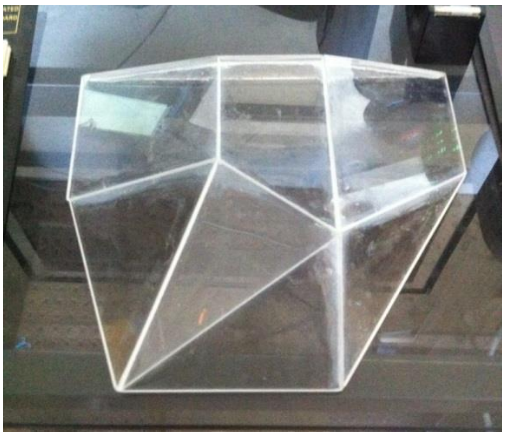
Figure 17, Silicone Used on Left Controller

The results of the silicone were good but it was not very strong. Using the silicone to hold the pieces together while applying the Acri-bond was a bad idea since the silicone separated the joins slightly. Later some of the pieces needed to be redone with Acri-bond. The silicone was stripped off the edges and the pieces were held together while the Acri-bond set for five minutes. Figure 18 shows the Acri-bond ready to be applied in a ventilated area. Glasses and gloves were required while sticking the pieces together.

The base of the left controller was sized and printed using Sketch-Up shown in figure 19. To size the base on the inside of the controller, first the thickness of the acrylic was added to the original model. The joystick's height was measured so that the clearance between the base and the top of the enclosure will fit the joystick. The base was then designed to sit at the joystick's height from the top of the controller with the edges sitting flush against the sides.

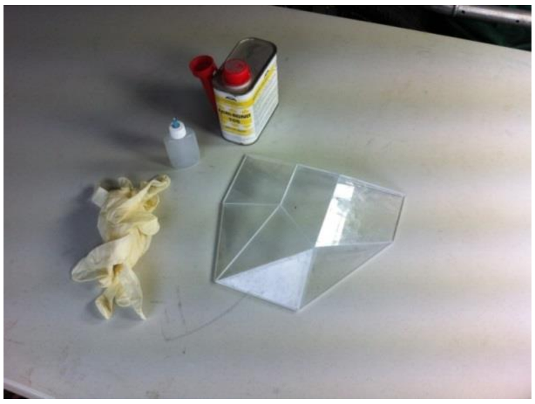
Figure 18, Ready to Apply Acri-bond

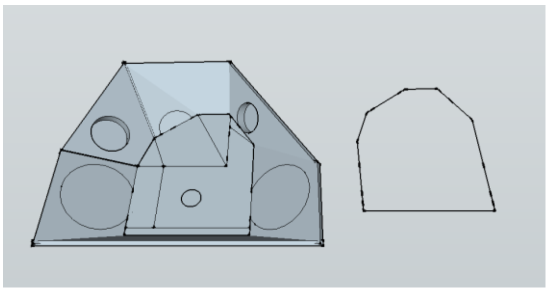
Figure 19, Design of the Base

The well fitted base can be seen in figure 20. The paper at the front is the template to cut the base. The orange acrylic pieces on the left were used to make the handle for the right controller. Figure 21 shows the controller with the Thumb joystick inside (later modified) and the bolt to hold the base plate. Later two more bolts were added to adjust the position of the base plate.

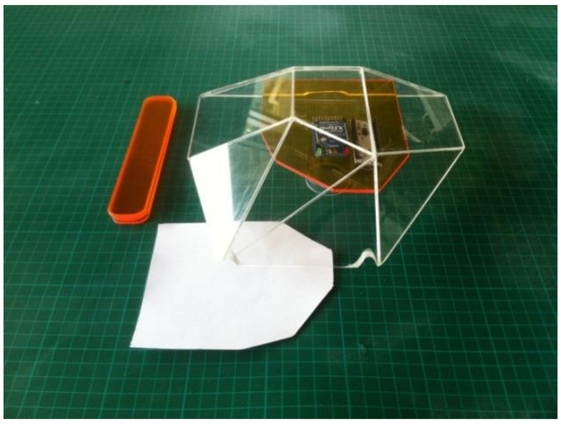
Figure 20, The Base showing some Components Inside

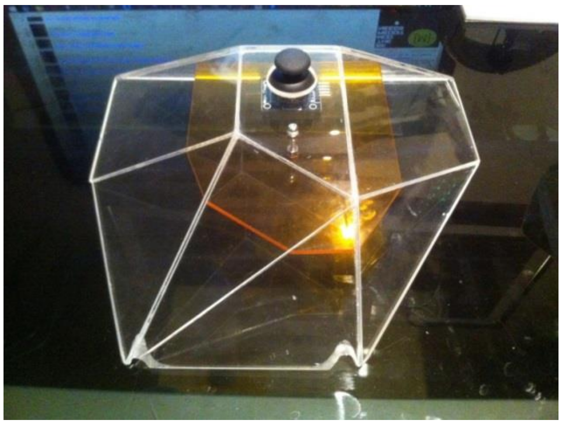
Figure 21, Joystick and Base Mounted

Figure 22 shows the left controller and the button numbering which will be consistent throughout this project. The numbering starts at ten because the 'grid of nine keys' are labelled as the first set of buttons in the Arduino program.

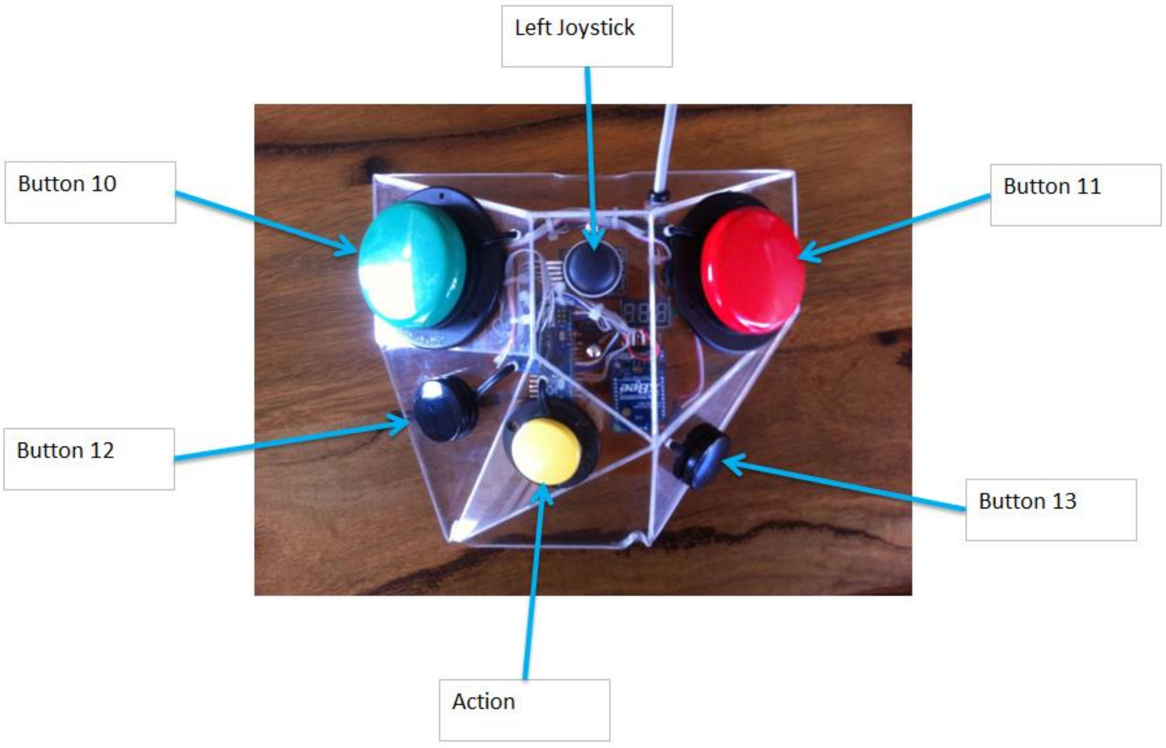

Figure 22, Left Controller with Buttons Assigned

The buttons used are from Ablenet. The large buttons have a face size of 2.5 inch (Ablenet, n.d.) and the small buttons are 3/8 inch in size (Ablenet, n.d.). The thumb joystick was ordered from Deals Extreme (DealExtreme, n.d.).

3.1.1. Modifications

After discussion about the usability of the thumb joystick, the thumb cap from the top of the left joystick was replaced with a stick instead of a dome for more accurate Joystick movements. This required a stainless wire to be placed in an existing hole of the stick (after removing the joystick cap) and hot gluing the new stick from a cheap remote controlled helicopter controller to the stainless wire. Stainless wire was used because it does not bend easily for small thicknesses. The orange base had to be lowered slightly for the new handle to fit since the joystick's cover sits a little higher. This also allowed more room for electronic components and the wiring. The modified left controller is shown with the stick joystick in figure 23. A voltage divider was later added (see figure 24) in order to be able to measure and then show the battery voltage on the display module.

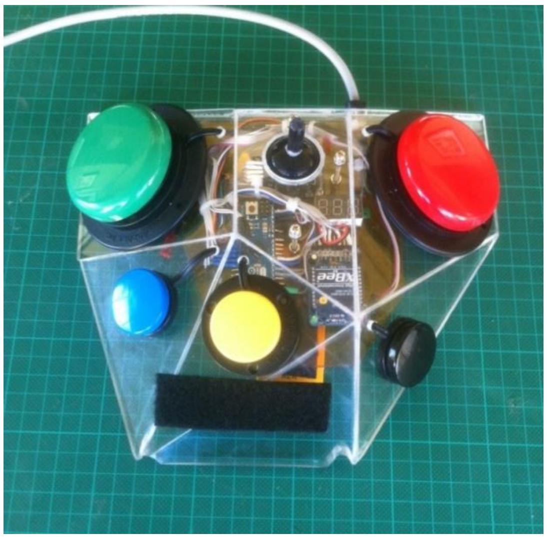
Figure 23, Different Joystick and Velcro Added

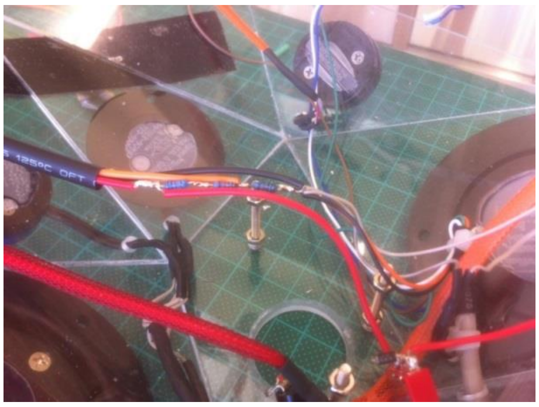
Figure 24, The Battery Voltage Divider

Figure 25 shows the neatest the wiring has been. Unfortunately the voltage divider in the large sheath had broken. This meant the sheath had to be cut back. When re-wiring the voltage divider two toothpicks were placed under the heat shrink so the resistor tails cannot bend and break. After that, the sheath was not re-installed.

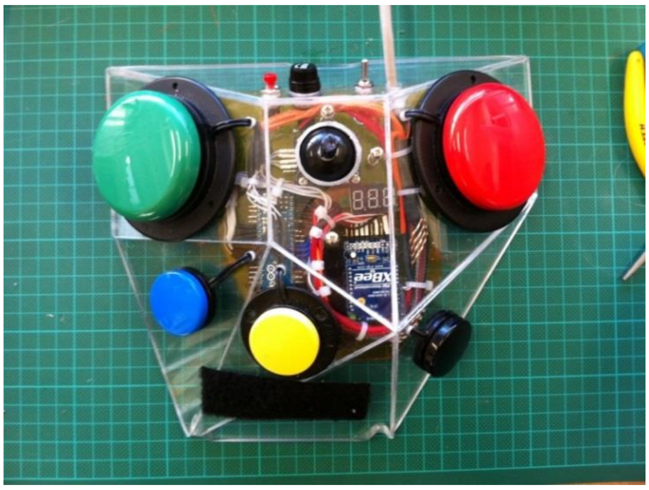
Figure 25, Neat Wiring Job

3.1.2. On the Chair

The figures in this section have the researcher sitting in the chair. Figure 26 shows the controller looking at the left arm rest. Figure 27 is viewing the front of the left controller showing the grooves that follow the arm rest. This helps lock the controller in place.

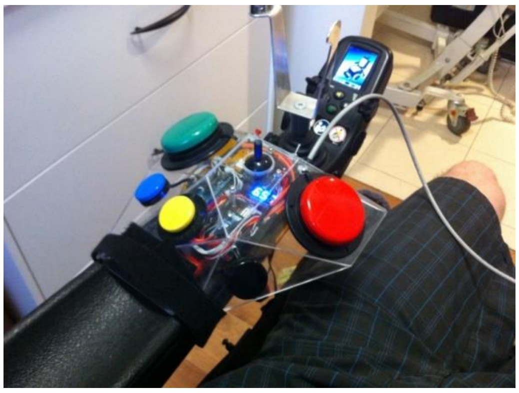
Figure 26, Left Controller in Arm Rest

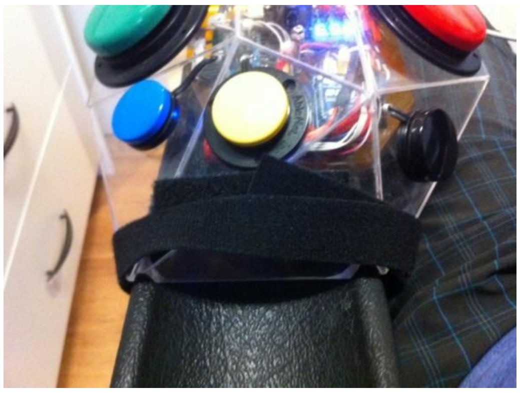
Figure 27, Left Controller Locked in Arm Rest by Grooves

Figure 28 shows how the left controller is supported at the back by another grove. The joystick is in reach by the user in the designed position shown in figure 29.

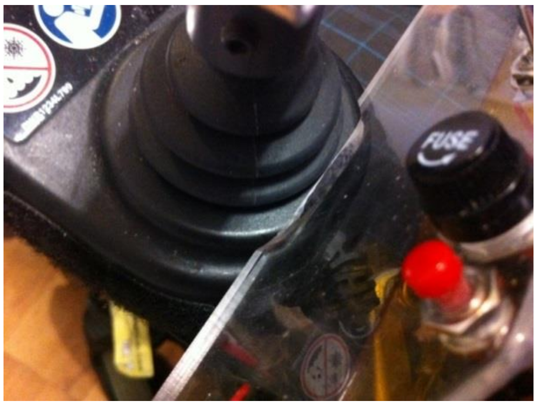
Figure 28, Back of Left Controller Locked onto the Chair's Joystick Base

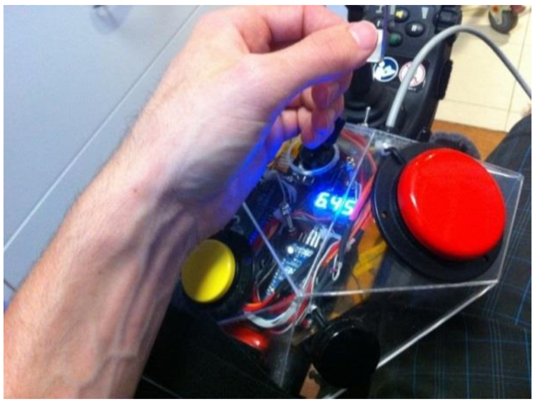
Figure 29, Joystick is reachable by the User (Showing the Researcher's Hand)

3.1.3. Problems Encountered

A hinge was going to be used to mount the plate to the enclosure. This would not have looked good and luckily ended up breaking the base. When drilling close to the edge of the first base plate, the acrylic snapped shown in figure 30. A new way to mount the plate was to use long bolts from the top of the controller.

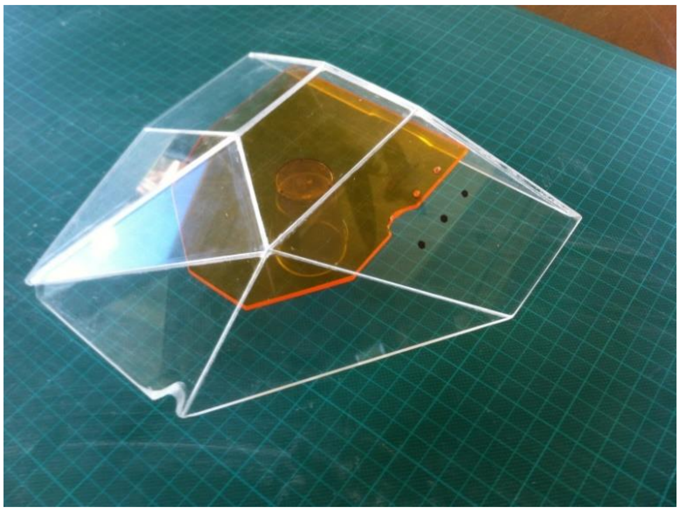

Figure 30, Drilling the Base Broke the Piece

3.2. Right Controller

The right joystick is a simple device which has a 2 axis Penny and Giles JC2000 Joystick (Penny + Giles, n.d.). The joystick selected is designed for disabled wheelchairs as shown in figure 31. This meant the spring tension that controls how hard to move the stick was a good match for the desired user. Also with Penny and Giles products they are designed so that different handles can be purchased. This made it easier to make a bush for the handle to fit because of the large thickness of the shaft. The cheaper joysticks have thin shafts.

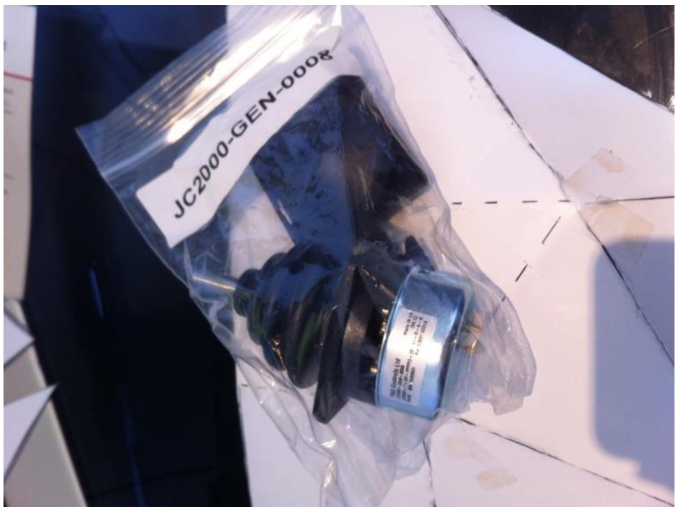

Figure 31, Right Joystick - Penny and Giles, JC2000

Figure 32 has the custom handle fitted. To build the handle a hole was drilled in the material and then countersunk while it was flat. Then the shape was formed by applying heat from a heat gun until the material was soft. The piece was then sandwiched between two metal strips where the bolt holds the handle before raising one of the ends. The other end was bent in the same manner.

The controller shown in figure 32 also has a button mounted to the right of the joystick. This button will be mainly used for the escape key and will be referred to as button 14. Figure 33 shows some of the internal wiring and connections. The box used is a mini ABS instrument case (Jaycar Electronics, n.d.). The box had just enough room for the joystick to fit and required a little bit of filing on the case where the plug fits into the bottom of the joystick. A seven wire shielded computer cable was used from the left to the right controller.

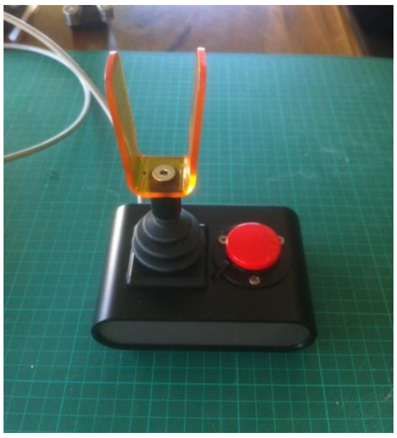
Figure 32, Right Joystick with Button 14

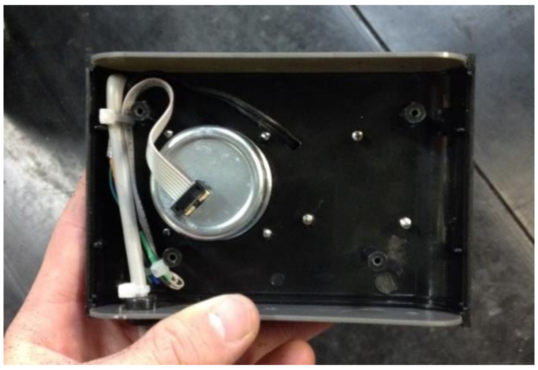
Figure 33, Inside Right Joystick

3.2.1. On the Chair

To illustrate the position and use of the handle figure 34 shows the joystick mounted in position with Velcro. Figure 35 shows how the user's hand locks into the handle.

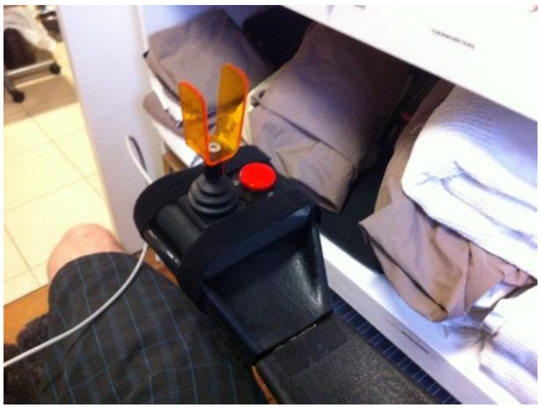
Figure 34, Right Controller fitted to Arm Rest

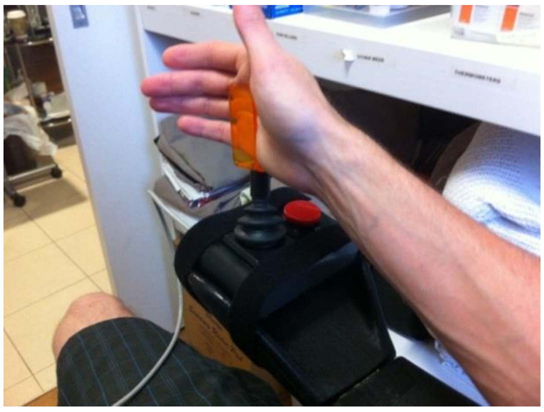
Figure 35, Right Joysticks Handle Locks the Users Hand (Researcher's Hand)

3.3. Display

The LCD Display houses the Xbee module, the micro Arduino and the LCD. A large 20 by 4 character LCD was needed to display a grid of 9 keys and a heading for the game profile. The contrast adjustment is really of benefit since every viewing angle requires a different contrast. Figure 36 shows the Xbee module, Arduino micro and the flat IDE cable installed in the enclosure. For the screw sockets to be installed, the corner plastic pieces needed to be filed down as indicated in figure 36 by the red arrows. This also allowed the LCD display to fit in the box since its corners were obstructing the sides.

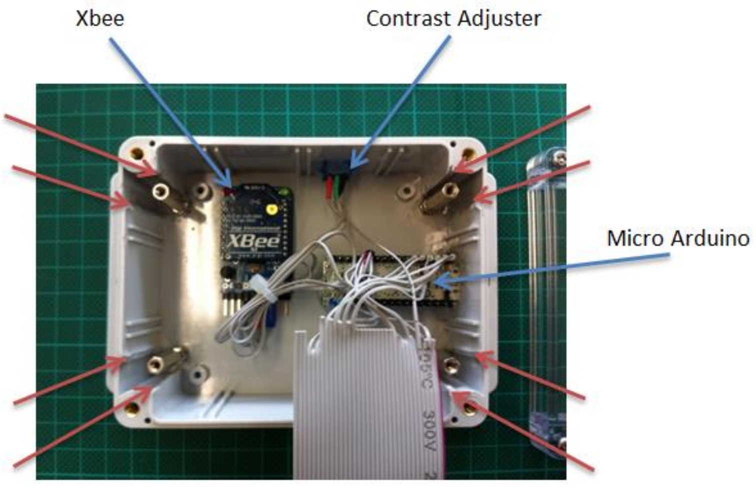

Figure 36, Inside the LCD Display

Figure 37 is a photo of the finished Display viewing the menu screen that appears at start-up. Figure 38 and figure 39 view the second and third menu pages. Figure 40 shows the format of the grid of nine keys when a profile has been selected. These screens can be navigated via the controller (See section 4.9).

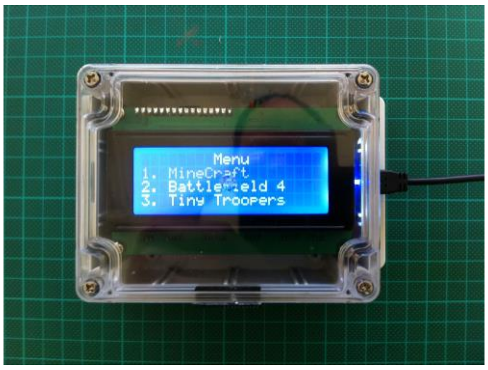

Figure 37, First Menu Page

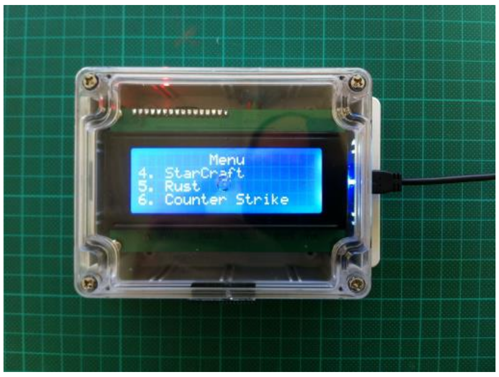

Figure 38, Second Menu Page

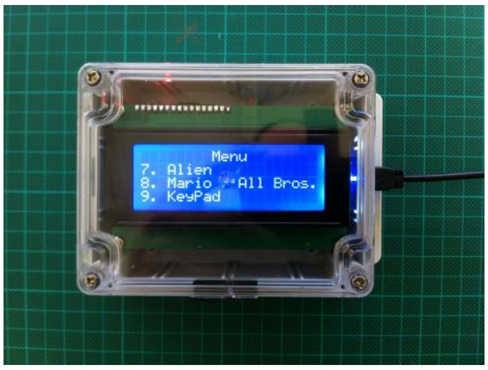

Figure 39, Third Menu Page

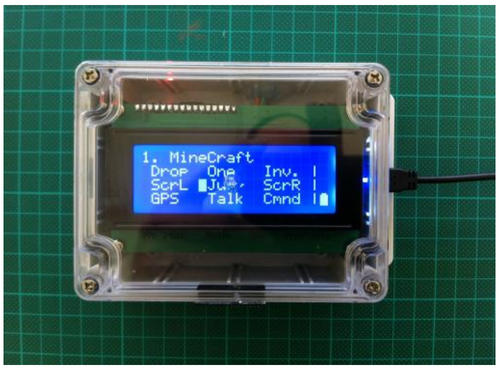

Figure 40, Minecraft Profile showing the Grid of Nine Key

4. Program Development Steps

For the first stages of programming, time was spent becoming familiar with the "Wiring" language used in the Arduino IDE (Arduino, n.d.). This programming language is related to C and C++ languages (Arduino, n.d.). This included using if statements and for loops along with Serial communication. Lots of these programs are available within the Arduino IDE. The examples are well documented and include the required hardware configuration. The Serial monitor was really helpful throughout the entire programming stage in order to extract a variable value at a given section of the program.

The following subsections list the order the programming was carried out in order to demonstrate the logical structure of the programming development.

4.1. The LCD

The LCD was quite simple to setup using the tutorial from Adafruit (Adafruit, n.d.) as they were well explained. They showed where each wire needs to go in the web page "Wiring a Character LCD" (Adafruit, 2012). There is a choice to connect eight or four data pins to the LCD, since the tutorial used the least number of data pins, four data pins was used. There are three control pins to be connected: EN, RS and RW. The EN pin controls when to read from the data pins and RS is used to decide if the data is a command for the LCD or a character to write (Adafruit, 2012). RW allows data to be read from the display, since this would not be valuable for this project it was tied to ground so that data is always written to the display (Adafruit, 2012). In total six digital pins were used for the LCD. Then the code to setup the display was given in the "Using a Character LCD" tutorial. The LiquidCrystal (Arduino, n.d.) library was used to set the cursor and print information to the screen from the Arduino. Later custom characters were created to display the battery level explained in section 4.10.

4.2. Xbee Setup and Configuration

The Xbees were setup initially using the program X-CTU from the Digi's download page (Digi, n.d.). To plug the Xbee into the computer an FTDI to USB cable (Sparkfun, n.d.) was used shown in figure 41. The FTDI socket can connect into the header of the Xbee adapter to be plugged into the computer by USB (Adafruit, n.d.). A picture of the adapter is show in figure 42.

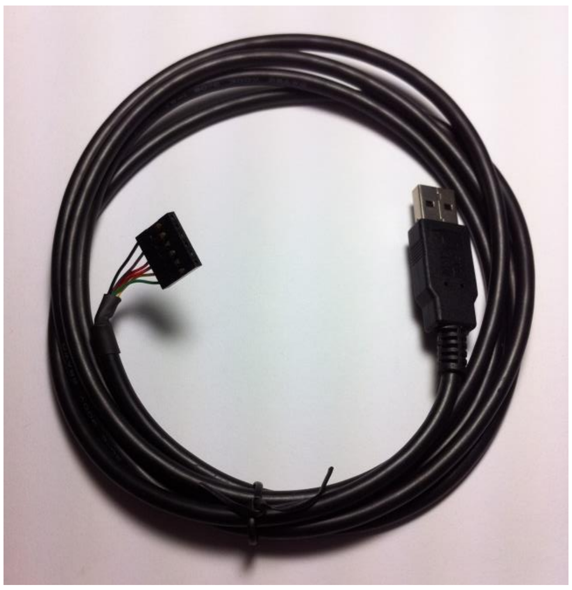

Figure 41, FTDI to USB Cable

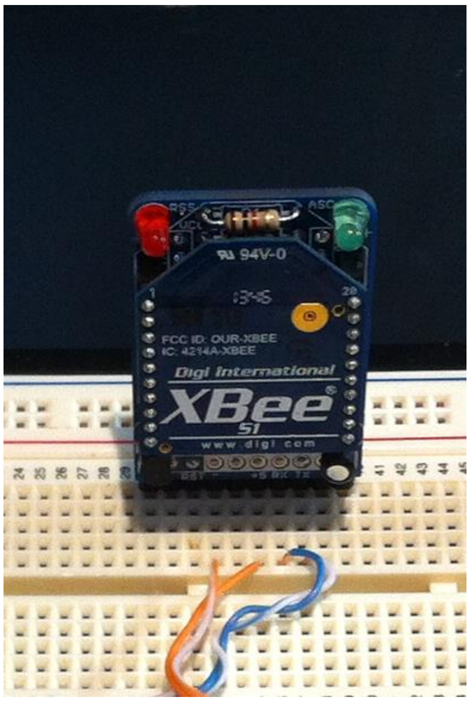

Figure 42, Xbee Adapter from Adafruit

The Xbees were setup to communicate using AT mode instead of API mode. API mode is a structured message (Digi, n.d.) whereas the AT mode (Digi, n.d.) is not. AT mode was used in this case for serial data transfer. From the cheat sheet provided by tunnelsup.com (Richee, 2012), one of the Xbees should be configured as an end point and the other, the coordinator. The end point reads information from the outside world and the coordinator has the task of controlling the network (Digi, n.d.). The Xbee used in the display was set as the coordinator since the power will be on in the display most of the time and the other was set as the end point since the standby option can only be used by the end device.

Once the Xbees were configured a serial program called Hercules (HW group, n.d.) was used to send text messages between computers through the USB's COM Port. This showed that the Xbees were working and there were no problems with the Xbee adapters. The adapter components were soldered to the board. Figure 42 shows the completed adapter. To solder the Xbee Adapters the tutorial by Adafruit was followed (Adafruit, 2012).

The Hercules program can be downloaded from HW group (HW group, n.d.).

The sleep mode of the Xbee was not setup until after the wires were soldered onto the Xbee adapter headers shown in figure 43. This meant the FTDI to USB cable could not be plugged into the adapter. To configure the sleep modes a different method was used shown in section 4.7.

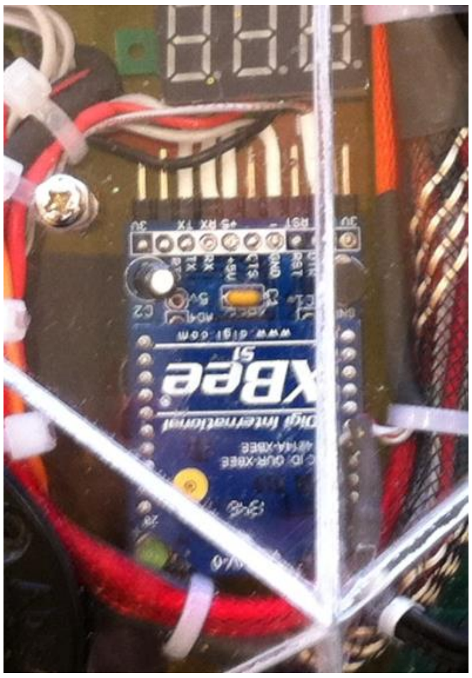

Figure 43, Xbee Adapter Soldered onto Headers

Software serial was chosen as the communication via the Arduino to the Xbee because it dedicates a choice of two digital pins for the communication. The Arduino micro has UART capabilities which could have been tested, but issues may have been raised because the pins that allow this communication are shared with the pins to upload the sketches and do button presses. Since software serial worked in the first instance of the program development this was chosen.

Software serial allows more than one serial device to be connected to a single Arduino. A limitation of the software serial is that the Arduino can only read from one device at a time (Arduino, n.d.). This did not cause a problem here because only the Xbee is connected for software serial communication. In the software serial example (Arduino, n.d.) in the Arduino 1.05-r2 version it explains the Arduino micro can use any pin to transmit and the pins 8, 9, 10, 11, 14, 15, 16 can be used for receive because they support change interrupts.

The software serial example (Arduino, 2012) was modified to be used for the serial communication to both Xbees. Pins 14 and 15 were used for both the display and the controller's software serial setup for simplicity and since the receive and transmit wires could be soldered into the corner of the board out of the way.

A mini program was done to test the wireless link. This was tested by sending a text message from the serial monitor of one Arduino to the LCD of the other Arduino. This mini program was later useful for debugging.

4.3. Reading Inputs

The output from the joysticks provides a zero to five volt reading which can be sent to the analogue input of the Arduino micro. The digital value from the analogue to digital converter is not so useful when something is to happen when the joystick is either left, right, up or down. To make the joysticks more useful the position of the joystick is converted into Boolean values which describe the position the joystick is in.

So if the joystick is in the top left position then the variables `JoyA_Left` and `JoyA_Up` will be one and the `JoyA_Right` and `JoyA_Down` will be zero. This task was done in the controller's Arduino since the display's Arduino needs to process key presses. In this way the Arduino in the display knows which position the joystick is in at any time and the information sent can be easily used as conditional tests in `if` statements. The switches were all switched to ground, therefore required the pull-up resistor which can be programmed when setting up the digital inputs by using `pinMode(pinNumber, INPUT_PULLUP)` (Arduino, n.d.). It was done this way to limit short circuits by not switching live wires. So Overall there are 8 Booleans for the two joysticks and 6 Booleans for the switches.

A test was performed to have all the switch inputs and joystick Booleans read by the controllers Arduino then sent to the display Arduino where the LCD screen will print the information as ones and zeros. These tests were done using a breadboard as shown in figure 44. The red cable that is partially hidden under the top of the breadboard and leaving the right side in figure 44 holds the analogue outputs simulating the left thumb stick which was still being shipped. The other end of the red cable goes to a joystick with the same characteristics as the thumb joystick. The grey computer cable entering from the bottom of the board is from the right joystick.

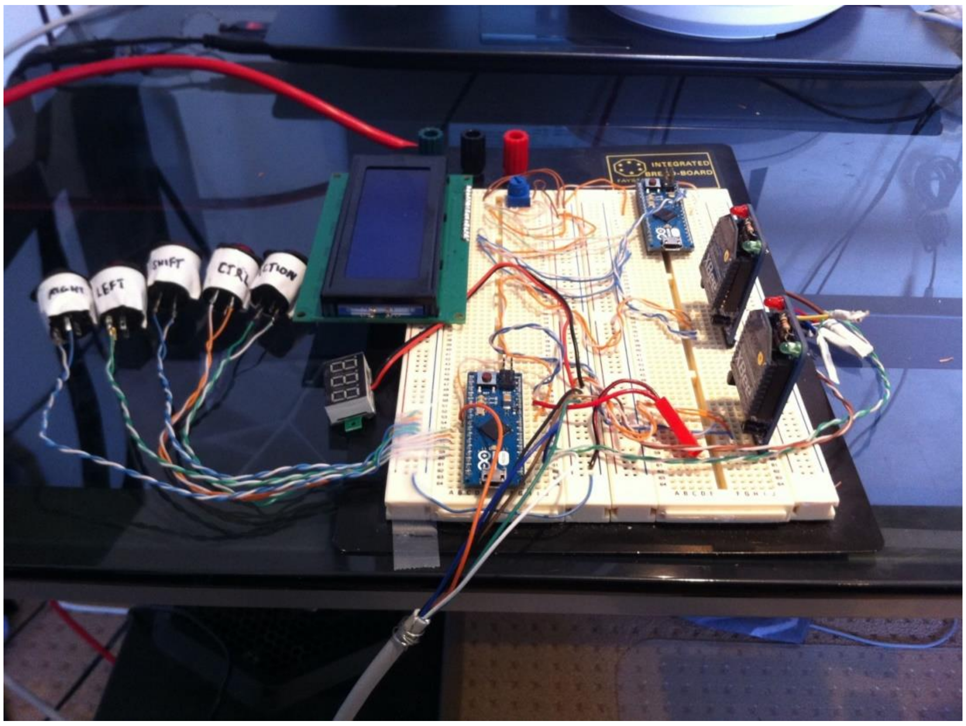

Figure 44, Testing the Components Together on the Breadboard

Each button and joystick Boolean was originally saved as an integer then transmitted. When declaring a variable as `int` the variable uses 16 bits but the Xbee can only send 8 bits? This caused no problems since the value of the integer never exceeded 255, the maximum value of a byte. Meaning only one byte was transmitted. For each byte of information sent only one bit was used out of the 8 bits available. This meant a lot of bits were sent which were not even used.

It was an advantage to condense all the Boolean information into one 16 bit integer, leaving two spare bits. This was done by using many `if` statements where the condition is the switch state. If the condition was true then `bitSet()` was used if not then `bitClear()` was used. Since the switches used pull-up resistors the bit was cleared when the switch was one. This made the integer value sent over the wireless link zero when no input was made by the user. The information was sent using `highByte()` and `lowByte()`. (Arduino, n.d.) Having the integer zero when the buttons and joysticks are not pressed was useful for putting the Xbee to sleep.

4.4. Using Millis()

At the stage where the joystick positions and the buttons could be read there was a requirement to program the screen to flash so that the user knows which key is selected in the grid of nine keys. If this was the only task running the flashing could have been implemented using `delay()`. However when delay is used, the program stops at the delay, for a period of time. This will not be ideal since other key presses will not work. The menu and the grid of nine keys both used a flashing display to indicate the selected key or heading. Later it was found an advantage to change the grid of nine keys to a blinking cursor next to the key in the display.

The function `millis()` was used instead of `delay()` (Arduino, n.d.), as the former is a counter of how many milliseconds the processor has been running using an unsigned long variable. This counter can go for 7.1 weeks without rolling over back to zero. When `millis()` is used with the modulus operator '`%`' in the Arduino IDE it was easy to set tasks to happen repeatedly. For example, to flash one of the menu headings' the code shown in figure 45 is used:

```
int rollover = millis() % 400;
if(rollover < 200){
  lcd.print("Heading");
} else {
  lcd.print("       ");
}
```

Figure 45, Using millis() to Flash a Heading

How the menu works will be explained in section 4.9. Figure 46 provides some sample code that controls the speed a loop executes. This is done by having an outer case that controls the execution speed of the loop. The hard part to this is selecting a small range of time. It should be large enough so that when the `if` statement is checked by the processor the condition of the if statement is true, but small enough so the condition happens once in the period selected. Otherwise the menu will skip the game profiles in the menu screen meaning the `profileselected` will increment twice in a short period of time.

For the code in figure 46 there is a 15 millisecond time frame for the code to execute once in a 800 ms period. This was selected by trial and error. Later it was realised that 15 milliseconds must be the scan cycle time. This code was an advantage for the user because the joystick can be held in one spot while the game profile selected slowly moves down/up the page. The condition to stop the `profileselected` from increasing outside its index range was emitted for simplicity.

```
if((millis() % 800) > 785){
  if(JoyADown == 1 || JoyBDown == 1){
      profileselected = profileselected + 1;
  }
```

Figure 46, Using millis() to move down Menu

Another place where millis() was used was to take a timestamp so that after a fixed time period has elapsed something can happen. This was used in the grid of nine keys to know when to move the blinking cursor back to the centre grid and also to navigate to the menu from the grid of nine keys by holding a button. These examples will be explained under the relevant headings.

4.5. Documentation

It was at this stage of the programming where the documentation was looking rather untidy and lacking explanation. So care was taken to write a comment on the lines of code that did not quite make sense so that the code could be understood next time it was visited. Also by splitting the code into sections with a heading explaining what the section does, created a program structure.

4.6. Grid of Nine Keys

The left joystick position was numbered from one to nine, one being the top left position, numbering from left to right, with three rows. With the four Booleans of the joystick the nine positions can be assigned using if…else statements. The position the joystick is in was saved as `grid`. To show the relationship between the joystick position and the grid see figure 47.

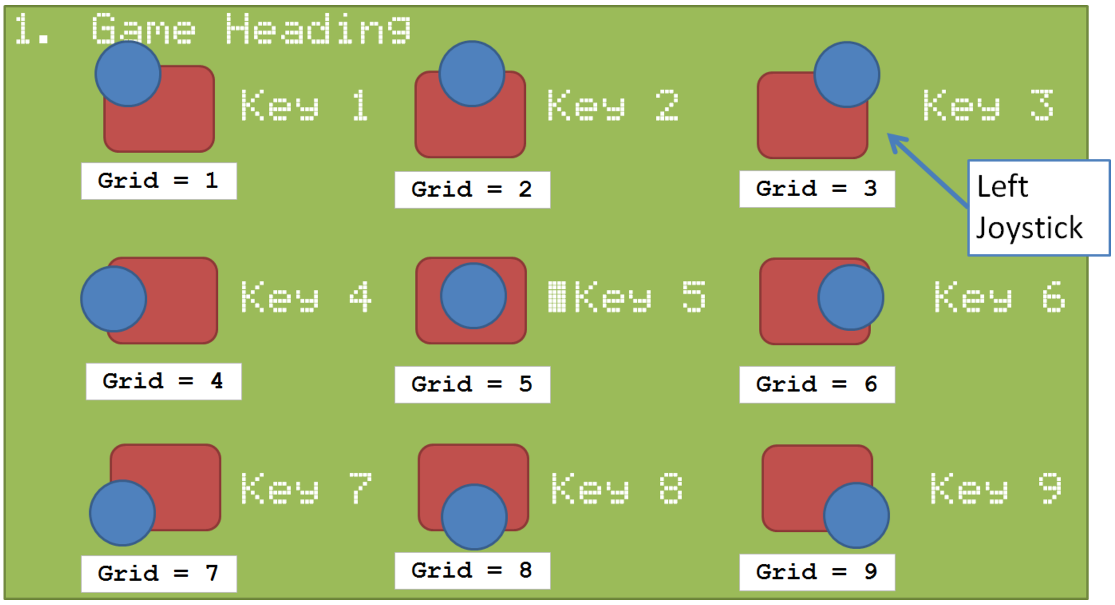

Figure 47, Joystick Position to Grid of Nine Keys

The order in which the if…else conditional tests were structured started with checking the corner grids, then the outer left, right, up, down positions followed by the centre position. It needed to be ordered so that the corner positions which need more conditions to be true are checked first, otherwise the left grid four will be on when the joystick is in grid one and grid sevens position. The condition for the `grid` to equal five was true if all the other statements were false.

Once the `grid` was mapped by the joystick's position, flashing of the selected key on the display needed to occur. For this a new `if...else` statement was created using the grid value instead of reading the joystick position again. It is clearer to read the program since the value of the grid controls the key press. Code was then inserted to flash the appropriate grid characters for the grid value selected via the joystick position.

There was a problem occurring in that if the joystick was moved while the LCD cleared the display, the previous grid would stay cleared. To fix this problem, the code where the screen was made to flash also had the grid number saved to a variable `action` as well. For example if the `grid` is one then save `action` as one, if the `grid` is two save `action` as two. By placing an `if` statement with the condition if: `action` is not equal to `grid` between the if...else statements previously mentioned; the conditional statement is true whenever the joystick has changed position. Ie. When the previous `grid` number (which is saved to `action`) is not equal to the current `grid` number then the grid has changed, allowing the screen to refresh. This condition was also useful for starting a timer so that the key will not return to the centre until a time has elapsed. This allows the screen to refresh only if the joystick has changed position. If the screen is made to refresh constantly the display could look dim because of the refresh rate of the screen.

Work was done on keeping the outer key flashing for a period of time so the action button can press that key. This was achieved by changing the condition of the last if statement that sets the grid value equal to five. By placing conditions that are false in this last statement the outer keys stay selected. To have the centre grid selected two conditions need to be checked:

1. Has the time from when the joystick was last changed elapsed?
2. Has the action button just been released?

The structure of the program was carefully chosen to ensure this worked properly.

A condition was that a variable `lockaction` must equal zero for the grid to equal five. Therefore when `lockaction` is one and the joystick is in the centre, the joystick will be blocked from returning the grid to the centre, hence a 'locking action' has occurred. The outer grids can be selected by the joystick even when `lockaction` equals one so that if a mistake was made the other key can be quickly selected again by the joystick. This will reset the time period for the grid to return back to centre.

Before the grid value is saved the negative trigger of the action button is checked as shown in figure 48. If the action button is pressed then `holdbutton` equals one, if the button is released then `releaseaction` equals one. Concentrating on the middle if statement in the example, if the button is held previously and has just been released then a negative trigger has occurred so we can make `lockaction` equal zero which lets the grid equal five.

```
if(val_action == 0){
  releaseaction = 1;
}
if (holdbutton == 1 && releaseaction == 1)
{
  lockaction = 0;
  holdbutton = 0;
  releaseaction = 0;
}
if (val_action == 1)
{
  holdbutton = 1;
  releaseaction = 0;
}
```

Figure 48, Code for the Negative Trigger

After this code the joystick position is mapped to the grid value. If a negative trigger has occurred and the joystick is in the centre then the grid will equal five. If not, a time from the change of the joystick position is checked.

If the joystick has changed two timestamps are saved. One is the timestamp when the joystick has changed position and the other is the same timestamp with the addition of a `dwelltime`. Thus if the `millis()` counter is currently in-between these two timestamps then lock the centre key from returning, meaning set `lockaction` equal to one.

When the `grid` value is changed to the centre from a negative trigger of the action button or the period of time elapsing from the last joystick change, the program would then detect the change in the `grid` value as a change in the joystick position (even though the joystick has not moved) and would set `lockaction` to equal one. This would then set the value of `grid` to the previous grid value and not allow the grid to equal 5 (returning back to centre). To allow the negative trigger to work the condition `grid` is not equal to five was added to the condition to set `lockaction` equalling one - this stops `lockaction` equalling one after a negative trigger sets `grid` to equal five. The purpose of this is to allow the centre grid to be selected when a time period has elapsed or a negative trigger of the action button occurs.

To stop the grid returning to the centre while the action button is pressed, before setting `lockaction` equal to zero the action button must be released. After this the centre key can be highlighted. Hence the action button can be held for as long as necessary before the grid returns back to centre. The code that sets `lockaction` is shown in figure 49.

```
// Change in joystick position has occured
if (action != Grid ){
  timeactuate = millis();
  timeactuatefive = timeactuate + dwelltime[profileselected];
}

if ( millis() >= timeactuate && millis() < timeactuatefive && Grid != 5){
  lockaction = 1 ;
}
else if(val_action == 0) {
  lockaction = 0;
}
```

Figure 49, How `lockaction` is set to Prevent the return to Centre Grid

Now there is a working display which flashes a certain grid selected by the joystick and does not return to the centre until a dwell time period has elapsed. The next step was to have the selected key pressed. Different key presses are controlled by using the `Keyboard.press()` function (Arduino, n.d.) placed in the code where the display is made to flash. The release of the key presses used in the grid of nine keys, happens near the start of the program and occurs on the condition if the 'action button' is not pressed.

4.7. Xbee Sleep

The controller can use less battery power by putting the Xbee to sleep when there is no input by the user. For the Xbee to sleep a new way to program the Xbee was needed since the wires were soldered onto the headers viewed in figure 43. After reading the product manual for the Xbee (Digi, n.d.) a command mode can be initialised (Arduino, n.d.) through the serial link of the Xbee.

The Xbee goes into command mode when it receives a +++ in its buffer without a carriage return (Arduino, n.d.). Then other commands can be entered to turn on sleep mode. In this way the wiring didn't have to be modified. To program this, the Arduino reads serial data from the serial monitor which was written to the Xbee in a simple program shown in figure 50.

```
//In the void loop part of the program:
if (xbee.available()){
  Serial.write(xbee.read()); //Returns the status of the Xbee
}
delay(100);
if (Serial.available()){
  xbee.write(Serial.read());
}
```

Figure 50, Serial Monitor to Xbee to Program Sleep Mode

Figure 51 has the sequence of commands needed to program sleep mode. These were found in the Xbee's product manual (Digi, n.d.). This needs to be typed into the serial monitor.

```
+++         // with no carriage return (CR)
ATSM2       // with CR     Sleep Mode 2 <50uA while sleep.
ATD83       // with CR     Set pin 9 (DTR/sleep_RQ) to digital input.
ATWR        // with CR     Write into non-volatile memory.
ATCN        // with CR     End Command Mode
```

Figure 51, Commands to turn on Sleep Mode

An extra wire was needed so that the Arduino can control when the Xbee sleeps. Pin 13, which turns on a green LED on the Arduino micro, was wired from the Arduino to the DTR pin of the Xbee adaptor. The green LED helped to see if the program was not setting the digital output on or if the Xbee was not sleeping. After successfully setting up the sleep mode, the pin was changed from pin 13 to pin 8 since the LED consumes the battery's power in the controller.

Now the Xbee can be put to sleep by setting pin 8's digital output high. An experiment was done to try putting the Xbee to sleep after each change of the controller's input. Initially the Xbee was asleep then a button was pressed, the Xbee then woke and a key press was simulated through the Arduino, the Xbee then went to sleep. The key would stay pressed until the release of the button which woke the Xbee to release the key, after that the Xbee slept again. This did not work, the Arduino simulated unwanted sporadic key presses. While debugging this problem, the entire message from the controller was read on the display, the message had unusual characters appended to the end of the message. The reason for this was not further investigated. The Xbee would be awake most of the time while playing games anyway because of the high demand of user input changes. Therefore the Xbee was controlled to stay awake for a second after no buttons and joysticks are pressed. This is to be sure the keys are released before sleeping the Xbee.

Before the integer number is set which holds the Boolean information of all the inputs, the previous numeric Boolean is saved as `previousnumericboolean`. When the new value is different to the old integer value a change has occurred with the inputs. When this happens a timestamp of `millis()` is saved which is used to sleep the Xbee a little time after there are no user inputs. The integer value is zero when there are no user inputs.

Figure 52 shows how the Xbee is put on standby and splits up the `numbericboolean` into bytes because that's what the Xbee sends. Here you can also see the integer Volts being transmitted. This was one method of sending the voltage information to the display. This method was used to log the voltage through the display module's serial port. `Volt` is the analogue to digital value of the voltage divider. This number gets processed in the display's Arduino to a floating point number.

```
if(numericboolean != previousnumericboolean){
  timechange = millis();
}
//numericboolean equals zero when nothing is pressed.
if((millis() > (timechange + tillstandby))&& numericboolean == 0){
  digitalWrite(standby, HIGH);  //Sleep
}
else
{
digitalWrite(standby, LOW);    //Write

//2.6 ms is needed to wake up from the sleep mode 2. (Digi, n.d.)
delay(5);

  xbee.write(":");
  xbee.write(lowByte(numericboolean));
  xbee.write(highByte(numericboolean));
  xbee.write(lowByte(volt));
  xbee.write(highByte(volt));
}
```

Figure 52, Sleep after One Second and Write to Xbee

To read from the display module's Xbee it is helpful to have a starting character to initialise the start of the message. The message can then be read by the code in figure 53.

```
//Save the first character as Bool
if (xbee.available())
{
  Bool = xbee.read();
}
if (xbee.available() && Bool == ':')
{
  lowboolean = xbee.read();
  highboolean = xbee.read();
  lowadcvoltage = xbee.read();
  highadcvoltage = xbee.read();
  if (xbee.available()) {
    //Removes characters from buffer
    while(xbee.available()) {
      xbee.read();
    }
  }
}
//Compiles the two bytes as an integer
numericboolean = word(highboolean,lowboolean);
```

Figure 53, Code to read Joystick, Buttons and Voltage from Xbee

Thus the controller will compress all the information into the binary digits of an integer number. The controller then transmits this number over the Xbee. After the message is received by the display, each bit of the integer is read using `bitRead()` and each Boolean is saved to a byte variable in the display where the variable can be used in if statements to control key presses.

4.8. Profiles

The different profiles should be able to display different keys to press for the grid of nine keys and be able to change the key or mouse click, to press for the other five buttons. This is required for different games that are controlled by different keyboard presses.

Therefore the Arduino micro needs to store information for each different customisable profile. For this arrays were used along with a menu system to index each array. The menu system will return a value that selects one of the profiles. The index `profileselected` then prints different strings to the display, different key presses and allows different things to happen. For example in the profile for the game “Rust” there is a requirement to hold the ‘u’ key while clicking a left mouse button. For this an array was made where the value one would return when the Rust profile is selected. New code can then be placed under the condition that when the new array indexed to the selected profile equals one a new section of the code is read. The same technique can be used to silence part of a program. This was useful if a release of a key was not meant to occur when a certain profile is selected. In this way the program can be modified without making changes to the original program. The downside is that the program becomes more complex.

4.9. Menu

For the menu system to function array information is saved into the Display Arduino as global variables. These arrays hold strings of keys to display on the LCD, keys to press and other Boolean arrays that control parts of the code allowing one profile to act differently to others. Then in the loop function the Xbee receives the data from the controller’s joysticks and switches, this is needed before entering the menu since the joysticks need to be read to move the selected profile. After this the program has two options based on the value of the `menuon` variable. It can either go into the menu, which it does on start-up because `menuon` is set as one, or it can go into the operating mode of the program. The aim of the menu is to return an index for the selection of the profiles. This index will then be used to return the desired key presses and characters to write to the LCD. The index is labelled `profileselected`.

To enter the menu from the operating mode it depends on the current profile. If the escape key is set to button 14, then when button 14 is held for 5 seconds the menu will open. If button 14 is not set as the escape key then holding button 13 for 5 seconds will allow the menu to open. The reason for this is that some games need a shift key to be pressed for a period of time while clicking the left or right mouse button. This occurs in the game StarCraft for example. The only way for the user to do this is to have button 14 as the shift key and then the left hand can be used to do the other mouse clicks and key presses. For this profile button 13 is set as the escape key so that the user can leave the game and also enter the menu to select different game profiles. This description is shown as a diagram in figure 54.

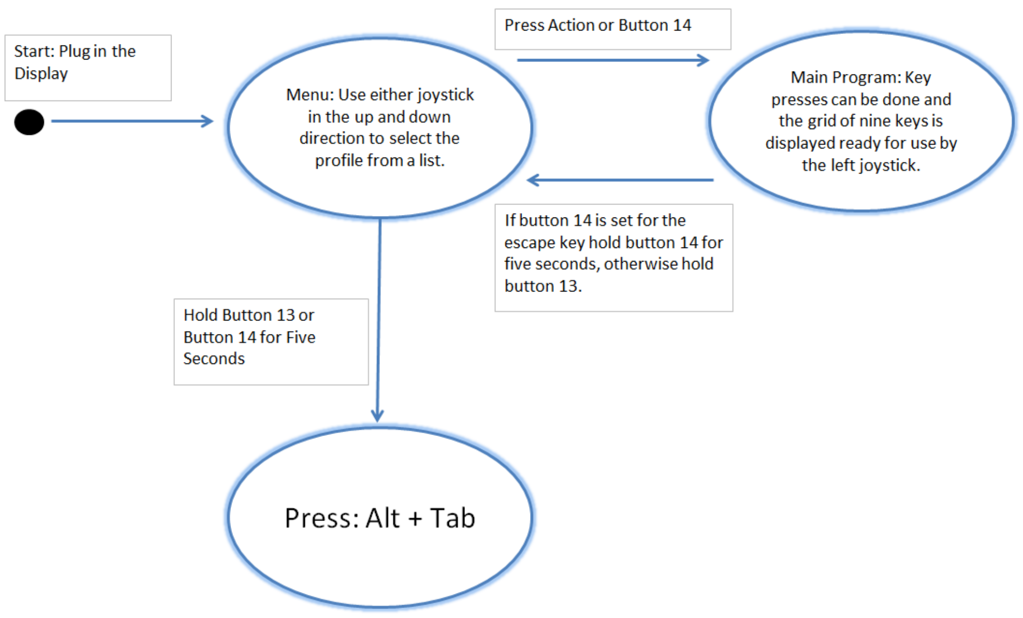
Start: Plug in the Display
Menu: Use either joystick in the up and down direction to select the profile from a list.
Press Action or Button 14
Main Program: Key presses can be done and the grid of nine keys is displayed ready for use by the left joystick.
If button 14 is set for the escape key hold button 14 for five seconds, otherwise hold button 13.
Hold Button 13 or Button 14 for Five Seconds
Press: Alt + Tab

Figure 54, Menu flow Diagram

To display the profile selected from a list, similar code to the way the grid flashes was used for the profile to flash. The `lcdpointer` is used to set the cursor on the correct line of the three lines currently available on the display. `Lcdpointer` is saved as `profileselected` modulo `3`. Then the Arduino will print the heading array indexed with the profile selected to that line.

To initialise the screen figure 55 explains how the LCD cursor is set and what information is printed. This happens every time the `profileselected` is incremented or decremented by the joystick.

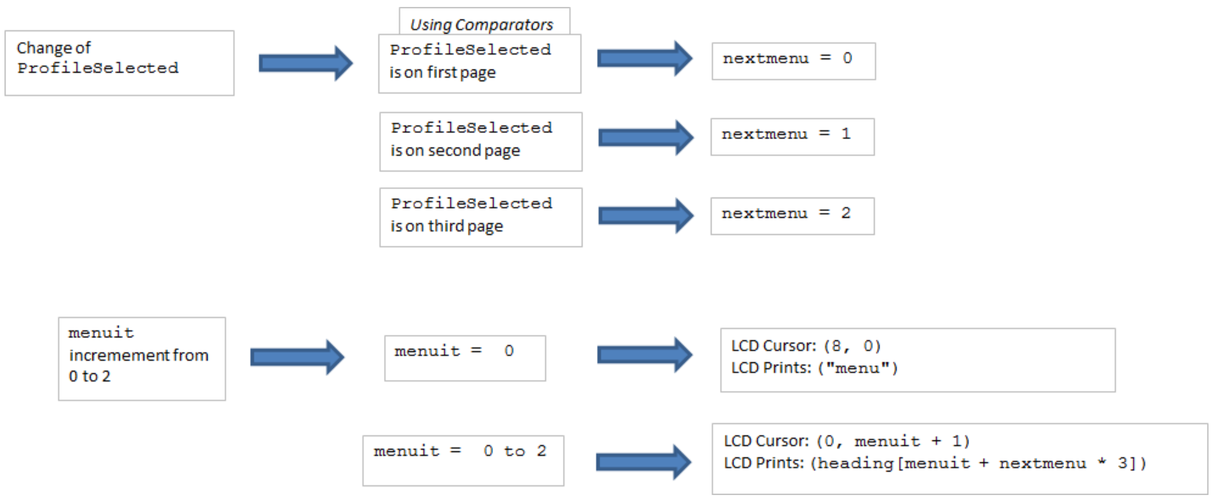
Change of ProfileSelected
Using Comparators
ProfileSelected is on first page
nextmenu = 0
ProfileSelected is on second page
nextmenu = 1
ProfileSelected is on third page
nextmenu = 2
menuit incremement from 0 to 2
menuit = 0
LCD Cursor: (8, 0)
LCD Prints: ("menu")
menuit = 0 to 2
LCD Cursor: (0, menuit + 1)
LCD Prints: (heading[menuit + nextmenu * 3])

Figure 55, Diagram of how the Menu Initialises

To display each line there is an incremented variable labelled `menuit` which stands for menu iteration. This value is used to refresh the screen. To refresh the screen `initmenu` is set to one. When `initmenu` equals one `menuit` is incremented from zero to two. When `menuit` equals zero three operations happen:

1. heading “menu” is displayed
2. The first profile heading is also printed.
3. Release of all keys from the keyboard and mouse occur.

The screen initialises when the `profileselected` is changed because, when selecting different profiles in the menu, the display may have written a blank line to the LCD.

When the menu exits the `refreshlcd` variable prints the grid of nine keys with the information of the profile selected. This is needed to write the profile heading on the top of the screen and because the grid of nine keys only displays information after a change of the joystick position has occurred. This means the menu screen would still appear after leaving the menu. A brief `delay()` is used in the menu section to allow the action or button 14 to be released before starting the operating mode.

Other interlocking variables were used to stop unwanted actions. Since the same button allows the menu to open and leave the menu, a method was required to stop the menu leaving straight after it was entered. To do this a variable `returnedfromprogram` is set to one upon entering the menu. `returnedfromprogram` is set to zero when both the action and button 14 are released. To leave the menu `returnedfromprogram` must equal zero. This means that the button to enter the menu while held down will not leave the menu until the action and button 14 has been released for `returnedfromprogram` to equal zero.

The menu was setup this way so that it is easier to add more game profiles. This was successfully demonstrated when three more profiles were easily added to the original six profiles that were originally set up.

4.10. Battery Voltage

The specification for the input Voltage limits is 6 – 20 Volts (Arduino, n.d.). A voltage divider was used to send a third of the battery voltage to the analogue input of the controller’s Arduino micro. This is to be able to use 15 Volts without exceeding the 5 Volt analogue input limit. For future works a higher battery Voltage may be used meaning the voltage divider will not need to be changed.

The Xbee can only send byte size information so the 16 bit integer from the anologue to digital converter was then sent using the `lowByte()` and `highByte()` functions. This integer was then concatenated on the display’s Arduino micro and scaled to the battery voltage using a floating point calculation. This was done to display the voltage and send serial data of the voltage for the graph in figure 57. It was later changed to send only a byte of information over the Xbee to the display which is used to indicate six different battery levels on the LCD. To scale the analogue to digital integer to voltage, figure 56 shows the calculation used.

```
adcvoltage = word(highadcvoltage, lowadcvoltage);

// converts to floating point variable.
fadcvoltage = adcvoltage;

// fadcvoltage is divided by 1024 to equal one at full scale
// 5 is the voltage at full scale
// A multiple of 3 is used to allow for the voltage divider
voltage = ( fadcvoltage * 5 * 3 ) / 1024 ;
```

Figure 56, Scaling Analogue to Digital Battery Voltage

A test was done to see how long the battery will last when the Xbee is constantly on. To do this the right joystick was positioned to constantly type a key. When this was done the Xbee stayed awake. The battery used in the test was two six Volt Nickel metal hydride batteries in parallel. The capacity of the combination gives 4000 milliamp-hours. Six Volts is the minimum voltage at which the Arduino can operate. Figure 57 shows the voltage read by the display every minute which was logged with the `millis()` counter to give the time the voltage was read. A tab delimiter was written in-between the `millis()` and voltage and this data was logged using a freeware RS232 data logger (Eltima, n.d.). This can be downloaded from Eltima Software (Eltima, n.d.).

The 'flow control' needed to be selected as none and 'append to file' was selected in case 'start logging' was accidently selected after the readings were taken, clearing the file. After the data was logged to a text file the data was copied to an Excel worksheet and the graph in figure 57 was produced using a ten point moving average for the voltage. The moving average was logged in red and the blue series is the actual Voltage. Two series were logged to show the error that increases as the input voltage diminishes to the minimum input voltage.

The test was finished when the voltage of the battery reached 6.05 Volts, measured using a multimeter. This was regarded as the end of the test since the analogue to digital converter could no longer display the correct voltage input to the Arduino micro at the minimum value of 6 Volts. As the battery voltage dropped below 6.1 Volts the voltage reading increased. This is shown as the spike in voltage at the end of the test.

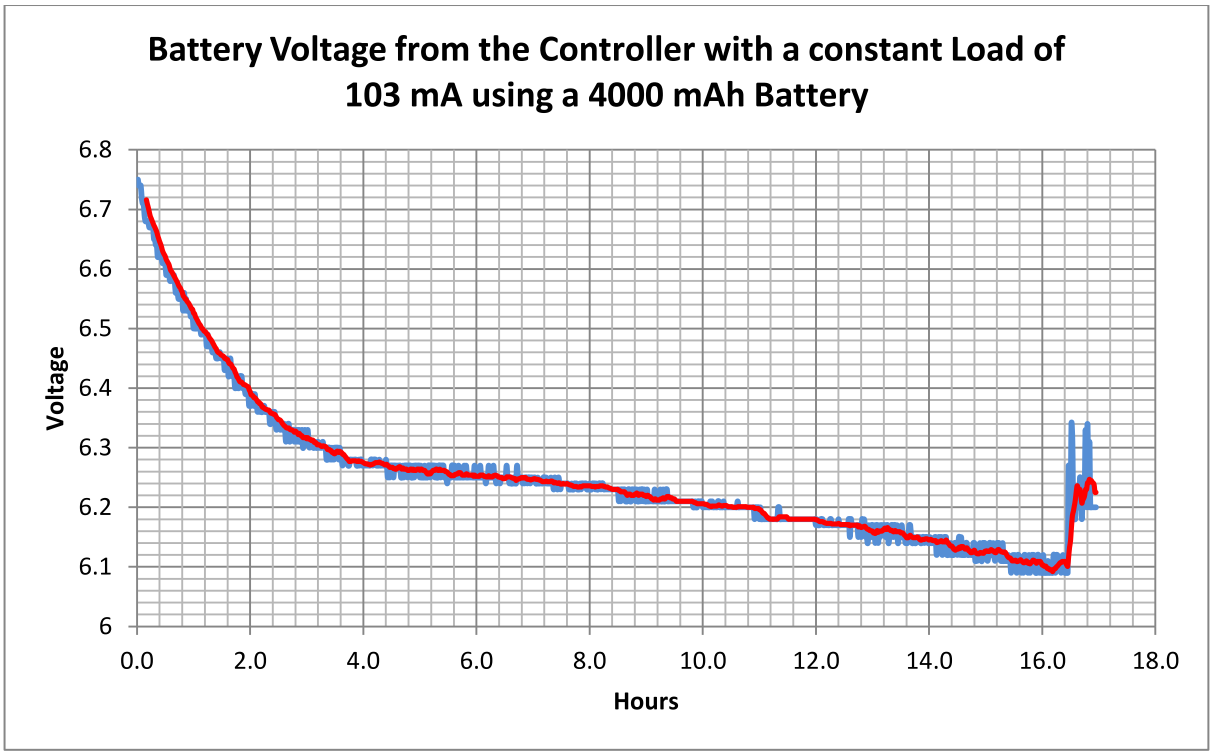

Figure 57, Log of Battery Voltage versus Hours of Operation

Regardless of the unusual spike in voltage the results were that the controller can be used constantly for 16 hours using the battery setup.

Displaying the actual battery voltage for the user was considered too much information, so a battery symbol was displayed instead that shows six levels of charge in the battery using six different symbols 0%, 20%, 40%, 60%, 80% and 100%. This meant the battery condition can be displayed in one character space of the display. To draw the symbol a program called a CGRAM designer tool was used (Arduino LCD module Custom Character Designer, n.d.). This sets up an array of bytes needed for the display. This program can be downloaded (Arduino LCD module Custom Character Designer, n.d.).

The `createChar()` function was used to store the symbols from the Arduino to the display where eight custom characters can be created (Arduino, n.d.). The Voltage reading took up 5 columns and the battery symbol uses 1, this meant the keys could be spread apart further so they are easy to read.

Since there is a spike in the voltage reading when the battery is flat this could indicate the incorrect battery voltage on the display. For this reason when the voltage falls below 6.11 Volts a variable holds the battery symbol low until the Voltage reading is greater than 6.4 Volts which is used to indicate a full battery charge. To increment the battery level sent to the display the voltage reading at every 3.2 hours was used from the graph in figure 57.

5. Wiring Diagrams

Final wiring diagrams were done after programming in case the wires needed to be changed for an unknown reason. A spread sheet was used during wiring the device to note where each wire is soldered. Figure 58 illustrates the wiring in the display module and figure 59 illustrates the wiring in the controller.

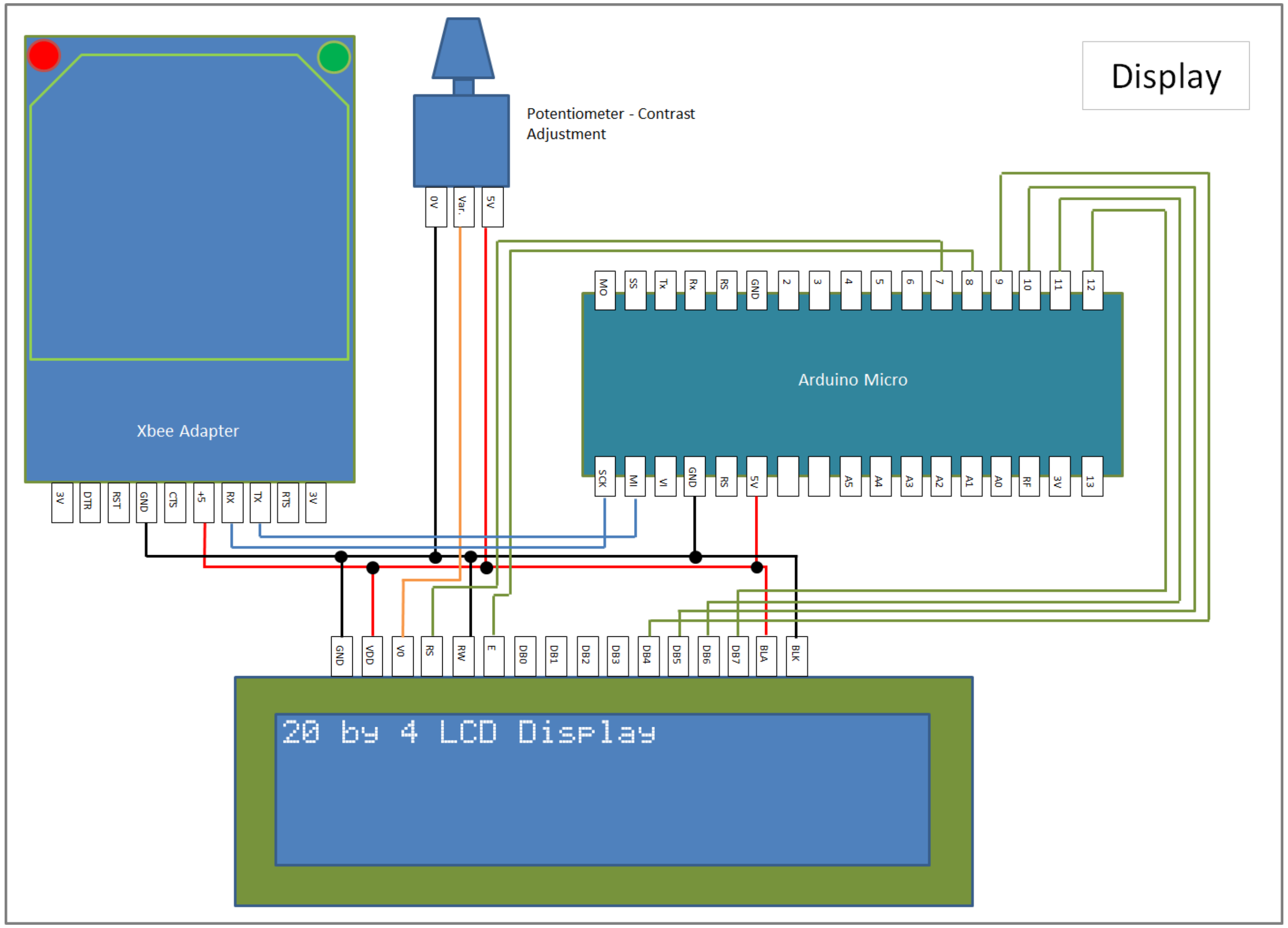

Figure 58, Display's Wiring Diagram

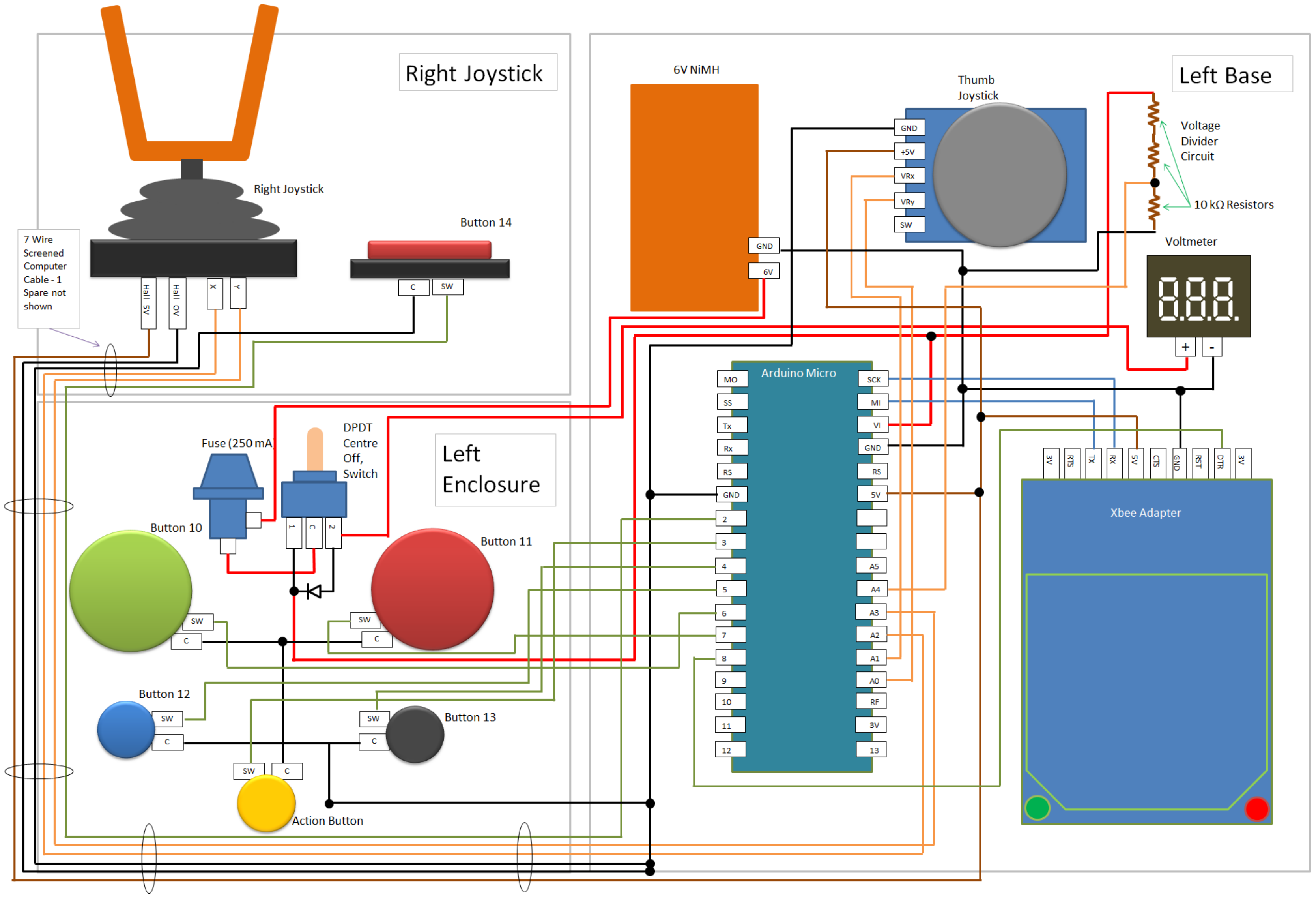

Figure 59, The Controller's Wiring Diagram

6. Test Results

The following questions were setup to validate the overall design of the controller. The interview was done after the outright approval from the ethics committee given on the 20th October 2014. The information and consent forms were filled in before the testing took place.

These results were evaluated with a flashing key on the grid of nine keys and had a voltage display instead of a battery symbol.

On average were you happy while using the controller?

Yes. I was keen to use it while work was being done on the controller.

Do you feel confident in the mechanical integrity of the enclosures?

Now that it has been modified, I do.

Does the functionality of the controller meaning the checking of the display to select a key lessen the impact or mood that the game is trying to create?

It doesn't because the display is in the peripheral vision of the computer screen. I also have to keep my head still for the Smartnav which keeps me in the game.

Was it hard to explain how to setup the controller to carers?

No, everybody managed it was simple. It was even better after having a power switch.

Did the battery last long enough for you to play the game properly?

Yes.

Is the wireless functionality a benefit over charging and replacing batteries?

Yes, so I can have a rest, move away and come back to the computer or move away for dinner without removing the controller from the chair.

Do you want to be able to use the controller in bed?

No, I worry too much about pressure areas on my skin. This leads to awkward positions anyway making the controller hard to use.

In what ways would it be possible for you to use the device in bed?

You'd have to be sitting right up which is uncomfortable and again pressure sores may be an issue.

Did you find that lots of rests were needed because of muscle soreness?

At first, my muscles got sore regularly. Then after time my muscles conditioned. I feel as though it is a good substitute for physiotherapy.

On average how long could you play games using the controller?

Four hours.

What sort of things spoilt your enjoyment?

I get confused with the key selections of the different profiles because I play a lot of games.

What custom designing would you do if another controller were made?

A gear stick knob on the end of the left joystick about the size of a Mentos made out of rubber.

Is social interaction a motive to play games?

Yes, it's good to communicate with gamers online.

Is the design of the 'grid of nine keys' too slow for most action games?

No, it seems to be quick enough. The outside keys on the grid aren't used as much during gameplay.

What made you stop using the controller?

Due to skin pressure relief I am not always in the chair, hence I can't play whenever I want.

How long did you spend playing games with this controller compared to other methods of playing games?

I was able to play with my mobile phone which has a touch screen game which was almost impossible then I realized how good it was when I could use the controller on the PC.

Was the whole time enjoyable?

I had some software problems with the computer. Not to do with the controller.

Was the limitation in most cases the way the game was designed or the hardware (controller)?

I thought the game design limited my abilities to play the game compared to the hardware.

Has the controller ever acted unusually or taken control of your computer?

It's never taken over the computer.

Do you have any issues with the response time of the controller?

I have no issues with the response time.

What changes (if any) would you like in the controller for future works?

For the key that flashes have the pixels that are lit for the character turn off and the rest of the pixels turn on, then alternate between the two conditions so that the key is always readable.

7. Input Lag

Input lag is the time taken from a button press to display the change on the monitor (Pasini, 2014). Figure 60 shows the processing time it takes the Micro Arduino to scan and write selected key presses for fifty data points.

7.1. Assumptions

This does not include the delay of the Personal Computer and the display, to process the images. This assumes that the message sent from the controller's Xbee is instantly received by the display's Xbee. It was noticeable that this was not the case when the Xbee sends the first message after the Xbee wakes from standby.

7.2. Results

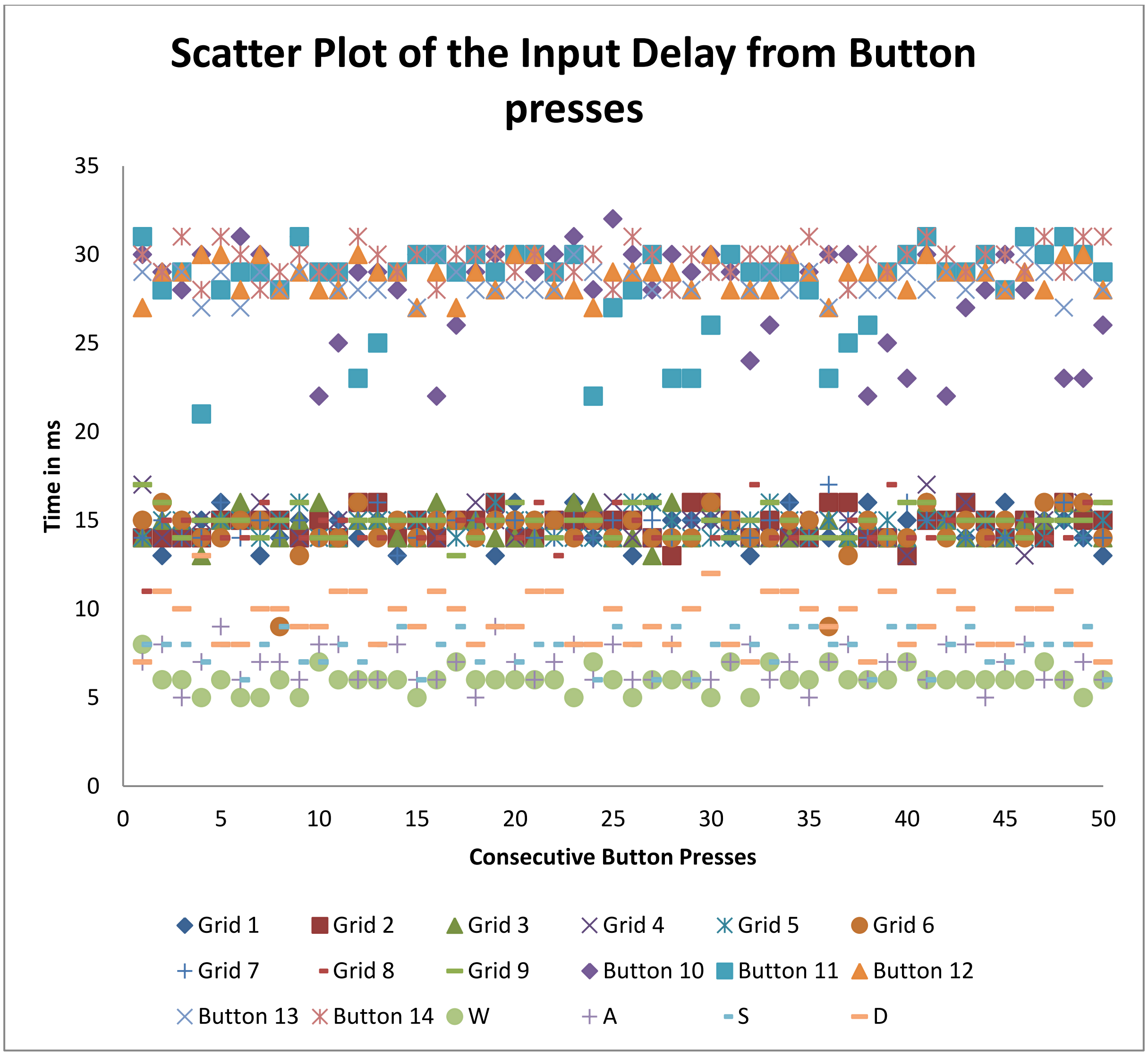

Figure 60, Input Delay from Button presses

The maximum processing time from the contribution of both Arduino microcontrollers is less than 32 ms. The scatter plot in figure 60 depicts that the time delay is relatively precise, with button 10 and 11 being the most variable. It should be noted that the W, A, S, D keys had the least processing times and variability because they were pressed at the top of the display program. By shifting when button 10 and 11 is pressed towards the top of the program could be an advantage for input lag. This is especially true when button 10 and 11 need a fast response from the player than the 'grid of nine key presses'.

7.3. Method

To produce the plot in figure 60 a separate copy of the program was saved and modified.

For the worst case of the delay to be measured two things would need to happen just after the program checks if the button is pushed:

- the button would need to be pressed;
- the timestamp saved.

The scan cycle of the controller would then need to do another scan before writing a message that the button is pressed, thus the delay would be the greatest time. If the button is pushed just before the processor scans the `if` statement for the button to pressed and the timestamp is saved, then this would create the best condition hence the shortest delay.

Therefore to compromise, the first timestamp was saved at the top of the program and the second timestamp before the message is sent. The serial monitor was used to see if the difference in delay was less than 255 (which it was). Then the difference in timestamps, which was a long variable, was cast (cast converts the variable type) to a byte which was then sent to the display (Arduino, n.d.).

In the display Arduino the first timestamp was taken at the top of the program again, this was just before the Xbee was read. The Xbee would then read the delay the controller took. In every instance of a keyboard press or mouse click the difference of the display's timestamp was added to the controller's difference. This was then printed to the serial monitor. This meant that the user can hold a button down and a list of time delays was printed in the serial monitor. This data was then copied to Excel to create figure 60.

8. Problems Encountered

8.1. Xbee Connectivity

After tidying the code for the display module there was an "ad" key press intermittently by the controller. It was later found that a delay of five milliseconds was accidentally deleted in the display's code. The delay was then re-written into the program and the problem was almost completely fixed. The following paragraph describes the steps taken to find the problem.

To check if the analogue to digital converter was unstable, which inadvertently pressed a key, or if there was an error in the message sent pin 13 will be set high when the analogue to digital converter senses the joystick has moved left. The action button will set digital pin 13 low again. The green LED on the Arduino micro will then turn on if the 'a' key is pressed via the joystick or by an unstable analogue to digital reading. The test showed the "ad" key press occurred without the green light coming on. This meant that there was undoubtedly a problem with the message sent.

A workaround was then tested because the end of the project completion date was close. To do this a checksum was included before the message is sent. The checksum is the sum of the data's values then converted to a byte where the number will be less than 255. Then the data is validated by the display's Arduino by calculating an additional checksum. If the checksum is not the same as the data checksum received then the previous values are rewritten to the variables that were read by the Xbee on the last successful transmission, also an error number will increment. This error number can be read by the serial monitor of the Arduino IDE by typing "errors" in the command window.

In addition to logging the amount of errors, when the error occurred all the values read from the Xbee were written to the serial monitor. Here it was easy to see that half of the message was received by the display with some other problems such as the format being incorrect or the message missing the first character, which is a ":".

While trying to find the problem a solution was to restore the Xbees to their factory settings and set them up on a new address and different channel in case it was interfering with something around it. To do this more commands were researched to setup the Xbee, these command are shown in figure 61 and were entered in between +++ and ATSM2 of figure 51.

```
ATRE        // with CR      Restores Defaults
ATID        // with CR      Displays the ID
ATID1000    // with CR      Sets the ID to 1000
ATCE0       // with CR      for end device, ATCE1 for Coordinator
ATSC1       // with CR      Sets the channel to channel 1
```

Figure 61, Commands to Restore the Xbee and Setup different Addresses and Channels

There were no advantages in restoring the Xbees and changing the channel.

The amount of times the error occurs was greatly reduced after adding a delay of five ms and the checksum will filter out the wrong messages if they occur. However, sometimes an error will occur when the display is first plugged in.

8.2. Ctrl, Alt and Delete Key Press

Prior to the Alt Tab function included in the menu system a Ctrl, Alt and Delete press was used. This was thought to be a good idea to be able to close programs which were not responding. The button press worked and the windows menu would appear to start the task manager. The problem is when Ctrl, Alt and Delete is pressed the SmartNav program stops working. This means that the mouse cursor cannot navigate to the task manager option to close the program that is not responding.

A solution was to silence this function, until later finding a better key press would be Alt and Tab which selects the desktop screen while in a game.

8.3. Batteries

The first battery ordered to be used in the controller was a three cell Lithium Polymer battery. These batteries are 11.1 V nominal but can be as high as 12.5 V after a charge. This higher voltage caused an unstable response from the analogue to digital converters in the Arduino, for the joysticks. Another problem with these batteries is that they go flat very quickly and without a voltage cut off, this unbalances the cells, which is not recommended. Lithium Polymers have a higher discharge current compared to Nickel metal hydride batteries, which can be dangerous if a short was to happen at the battery terminals. This could cause a fire. Though a fuse was added to the controller there is still a chance a short circuit can happen on the battery leads entering the base. Due to these dangers a Nickel metal hydride battery was used for this project which has a lower discharge current.

8.4. Corner Keys

The thumb joystick was too responsive making it hard to highlight the corner keys. For example when the top left grid key is to be pressed then the user will push the joystick to the top left, here the up and the left Boolean variables are true. Now the user releases the joystick. On release if the processor senses the joystick is in the up position and the left Boolean is false then the grid will change to the top grid. If the processor senses the left position on release without the up position, then the left grid will be highlighted. To fix this problem the joystick was read every 50 ms in a half second period timed using the function `millis()` (Arduino, n.d.). This responsiveness can also be adjusted per profile.

9. Future Improvements

9.1. Xbee Parameters

From the problem described in section 8.1 more research needs to be done to remove any remaining errors.

9.2. Customisation of the Profiles

Each profile needs a unique formula of key presses and some need key and mouse combinations. This makes it hard for each grid to be customizable for all the possibilities that might be needed to create the perfect profile for the user. Though all the possibilities would be hard to program it would be of benefit to finish the framework started in the top section of the program where the profile information is stored. Some of the arrays in the top section of the program are configured as a selector, for example to enable a ctrl key press with a keyboard press. This can be configured by writing in the correct digits in the part of the array where that profile belongs. It was intended for this to happen for each key but was not finished for all grid numbers because of the complexity of the programming required and time constraints for this project.

9.3. The Left Enclosure

Though the enclosure is aesthetically pleasing due to the transparent material displaying the components, another possibility which was thought of after cutting the acrylic sheet into pieces was using a 3d printer to design the outer case. This could then be made to have interlocking pieces which could be glued together making the enclosure much easier and quicker to build. A 3d printer can print in different colour materials. These colours could then suit the user's requirements.

10. Conclusion

The controller would not have worked without the continuous meetings with the user in the early stages of the design to determine the functionality of the controller. The early stages were the most difficult because it was uncertain what barriers lay ahead. These being physical barriers with arm movement, mechanical problems and problems with the electronic modules bought.

The enclosure imposed problems which required more rework than expected. The extra work and time put in to the enclosure created a more aesthetically pleasing device, but the time could have been used to:

- decrease power consumption in the controller;
- finish programming each grid of nine keys and buttons to be fully customizable;
- reducing input delay.

The controller can be used to play a variety of games when using a mouse tracker.   The custom design of the enclosure to suit the user made all the buttons, joysticks and switches reachable. The compact design meant that the controller did not interfere with food trays being placed on his lap and he was able to go outside without the controller hitting the doorway.

Using the display to press nine possible keys was an advantage so that lots of different key presses can be done through a joystick and button. This helped make the enclosure smaller and allowed the button faces to be larger than the regular sized key of a keyboard.

The right controller activates the game characters' movement and was signalled using the conventional key presses being the W, A, S, D keys. The escape key on the side of the right controller was a good placement since the user stops moving the player when wanting to quit the game. The additional four keys that do a single key press or mouse click were an added benefit in the game where a quick action needed to be done, compared to the grid of nine keys.

The menu was a way to change which keys were pressed for the different games being played. It was also a benefit to have the user control the menu through the controller.

The controller has opened a new avenue of entertainment to the user to:

- learning about upcoming games;
- watching videos on new strategies to play games;
- upgrading the hardware for the best effects;
- talking to others while playing games.

All these aspects create a zest in the user which made building the game controller worthwhile.

11. References

Plugin: Unfold Tool. (2007, June 08). Retrieved October 28, 2014, from My Sketch-Up Plugins list...: http://sketchuptips.blogspot.com.au/2007/08/plugin-unfoldrb.html

Ablenet. (n.d.). *Connections Strategies*. Retrieved August 07, 2014, from Ablenet: http://www.ablenetinc.com/Assistive-Technology/Switches

Ablenet. (n.d.). *Jelly Bean Twist-Top Switch*. Retrieved October 29, 2014, from Ablenet: http://www.ablenetinc.com/Assistive-Technology/Switches/Jelly-Bean

Ablenet. (n.d.). *Specs (R) Switch*. Retrieved October 29, 2014, from Ablenet: http://www.ablenetinc.com/Assistive-Technology/Switches/Specs-Switch

Adafruit. (2012, December 14). *Assemble your kit*. Retrieved November 18, 2014, from Xbee Radios: http://www.ladyada.net/make/xbee/solder.html

Adafruit. (2012, July 29). *Wiring a Character LCD*. Retrieved November 18, 2014, from adafruit: https://learn.adafruit.com/character-lcds/wiring-a-character-lcd

Adafruit. (n.d.). *Learn Arduino*. Retrieved December 22, 2014, from Adafruit: https://learn.adafruit.com/category/learn-arduino

Adafruit. (n.d.). *Standard LCD 20x4 + extras*. Retrieved August 07, 2014, from adafruit: http://www.adafruit.com/products/198

Adafruit. (n.d.). *Xbee Adapter kit - v1.1*. Retrieved November 4, 2014, from adafruit: http://www.adafruit.com/product/126

Arduino. (2012, May 25). software serial multiple serial test. *Arduino 1.05-r2*. Tom Igoe.

Arduino. (n.d.). *Arduino Micro*. Retrieved August 07, 2014, from Arduino: http://arduino.cc/en/Main/arduinoBoardMicro

Arduino. (n.d.). *Arduino Xbee Shield*. Retrieved September 18, 2014, from Arduino: http://arduino.cc/en/Guide/ArduinoXbeeShield

Arduino. (n.d.). *Cast*. Retrieved November 22, 2014, from Arduino: http://www.arduino.cc/en/Reference/Cast

Arduino. (n.d.). *createChar()*. Retrieved November 12, 2014, from Arduino: http://arduino.cc/en/Reference/LiquidCrystalCreateChar

Arduino. (n.d.). *Digital Pins*. Retrieved November 05, 2014, from Arduino: http://arduino.cc/en/Tutorial/DigitalPins

Arduino. (n.d.). *Guide to the Ardunio Leonardo and Micro*. Retrieved October 29, 2014, from Arduino: http://arduino.cc/en/Guide/ArduinoLeonardoMicro?from=Guide.ArduinoLeonardo

Arduino. (n.d.). *highByte*. Retrieved November 4, 2014, from Arduino: http://arduino.cc/en/Reference/HighByte

Arduino. (n.d.). *Language Reference*. Retrieved December 22, 2014, from Arduino: http://arduino.cc/en/Reference/HomePage

Arduino. (n.d.). *LiquidCrystal Library*. Retrieved November 2014, 2014, from Arduino: http://arduino.cc/en/Reference/LiquidCrystal

Arduino. (n.d.). *millis()*. Retrieved September 12, 2014, from Arduino: http://arduino.cc/en/Reference/Millis

Arduino. (n.d.). *Mouse and Keyboard libraries*. Retrieved August 07, 2014, from Arduino: http://arduino.cc/en/Reference/MouseKeyboard

Arduino. (n.d.). *SoftwareSerial Library*. Retrieved December 22, 2014, from Arduino: http://arduino.cc/en/Reference/softwareSerial

Arduino. (n.d.). *What is Arduino?* Retrieved December 22, 2014, from Arduino: http://arduino.cc/en/guide/introduction

ATA. (n.d.). *ACRI-BOND 105*. Retrieved September 10, 2014, from http://www.acrylictech.com.au/Acri-Bond-105.html

Broadened Horizons. (n.d.). *Ultimate Arcade 2 Limited Dexterity ...* Retrieved August 07, 2014, from http://www.broadenedhorizons.com/ultimate-arcade2-split

Davison, F. (n.d.). *A game controller for quadriplegics*. Retrieved Novemeber 22, 2014, from QuadStick: http://www.quadstick.com/

DealExtreme. (n.d.). *Keyes Thumb Joystick Module for Arduino*. Retrieved October 29, 2014, from DealExtreme: http://www.dx.com/p/repair-parts-replacement-analog-stick-module-for-ps2-controller-black-121340#.VFBYtqOQ-70

Digi. (n.d.). *The AT Command Set*. Retrieved November 17, 2014, from Digi: http://www.digi.com/support/kbase/kbaseresultdetl?id=2205

Digi. (n.d.). *What is API (Application Programming Interface) Mode and how does it work?* Retrieved November 17, 2014, from Digi: What is API (Application Programming Interface) Mode and how does it work?

Digi. (n.d.). *Wireless Mesh Networking.* Retrieved December 22, 2014, from Digi: http://www.digi.com/pdf/wp_zigbeevsdigimesh.pdf

Digi. (n.d.). *Xbee (R) 802.15.4 Device connectivity using multipoint wireless networks*. Retrieved September 18, 2014, from Digi: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-series1-module#docs

Digi. (n.d.). *X-CTU (XCTU) Software*. Retrieved October 15, 2014, from Digi: http://www.digi.com/support/kbase/kbaseresultdetl?id=2125

Dockery, S. (2011, June). *With unique mouth controllers, quadriplegic gamers can play*. Retrieved August 07, 2014, from The Bulletin: http://www.bendbulletin.com/news/1418278-151/with-unique-mouth-controllers-quadriplegic-gamers-can-play#

Eltima. (n.d.). *Capture serial port activity*. Retrieved November 12, 2014, from Eltima Software: http://www.eltima.com/products/rs232-data-logger/

Entertainment Software Association. (2014, April). *2014 Essential Facts about the Computer and Video Game Industry.* Retrieved August 07, 2014, from http://www.theesa.com/facts/pdfs/esa_ef_2014.pdf

Forth Inc. (n.d.). Retrieved October 29, 2014, from Forth Inc.: http://www.forth.com/

Gee, J. P. (2005). *Why Video Games are Good For Your Soul.* Australia: Common Ground Publishing Pty Ltd.

HW group. (n.d.). *Hercules SETUP Utility*. Retrieved November 17, 2014, from HW Group: http://www.hw-group.com/products/hercules/index_en.html

Jaycar Electronics. (n.d.). *Mini ABS Instrument Case*. Retrieved October 13, 2014, from Jaycar: http://search.jaycar.com.au/search?w=HB5960&view=list

Murdoch University. (n.d.). *Applying for human research ethics approval*. Retrieved September 12, 2014, from Murdoch University: http://our.murdoch.edu.au/Research-Ethics-and-Integrity/Human-research-ethics/Application/

Apparelyzed. (n.d.). Retrieved 12 19, 2014, from Quadriplegia and Quadriplegic: http://www.apparelyzed.com/quadriplegia-quadriplegic.html

Arduino LCD module Custom Character Designer. (n.d.). Retrieved November 12, 2014, from http://www.vwlowen.co.uk/arduino/wizard.htm

National Instruments. (2006, September). *Using LabVIEW to Simulate keyboard Events*. Retrieved October 27, 2014, from National Instruments: http://www.ni.com/example/28711/en/

Pasini, F. L. (2014, February 10). *The Myths Of Graphics Card Performance: Debunked, Part 1*. Retrieved November 21, 2014, from Tom's Hardware: http://www.tomshardware.com/reviews/graphics-card-myths,3694-4.html

Penny + Giles. (n.d.). *Muti-Axis Contactless Joystick Controller*. Retrieved September 10, 2014, from Penny + Giles: http://www.pennyandgiles.com/Products/Joystick-Controls/Multi-Axis-Contactless-Joystick-Controller-JC2000.aspx

Quad Control. (n.d.). Retrieved November 22, 2014, from QuadControl: quadcontrol.com

Richee. (2012, November 30). *Xbee S2 Quick Reference Guide/Cheat Sheet ...* Retrieved November 3, 2014, from TunnelsUp: http://www.tunnelsup.com/xbee-s2-quick-reference-guide-cheat-sheet/

SmartNav. (2012, August). *What is SmartNav?* Retrieved July 30, 2014, from SmartNav by NaturalPoint: https://www.naturalpoint.com/smartnav/

Sparkfun. (n.d.). *FTDI Cable 5V VCC-3.3V I/O*. Retrieved November 05, 2014, from Sparkfun: https://www.sparkfun.com/products/9717

SparkFun. (n.d.). *Xbee 1mW Trace Antenna - Series 1*. Retrieved August 07, 2014, from SparkFun: https://www.sparkfun.com/products/11215

SparkFun. (n.d.). *Xbee Buying Guide*. Retrieved October 28, 2014, from SparkFun: https://www.sparkfun.com/pages/xbee_guide

The AbleGamers Foundation. (2014). Retrieved August 7, 2014, from the ablegamers foundation: http://www.ablegamers.com/

Trimble. (n.d.). *Angle Between Faces*. Retrieved October 29, 2014, from Extension Warehouse: https://extensions.sketchup.com/en/content/angle-between-faces

Trimble. (n.d.). *Hobbyists, kids and...* Retrieved October 29, 2014, from SketchUp: http://www.sketchup.com/products/sketchup-make

tunnelsup. (2015, January). *Xbee Basics - Lesson 1 - General Information and Initial Setup*. Retrieved October 28, 2014, from Youtube: https://www.youtube.com/watch?v=odekkumB3WQ