University Of Westminster

Computer Science ECSC699

Electronics and Computer Science

Gesture Controlled Robotics

Author:
W1387769 Mohammed
Rahim Baraky

Supervisor:
Dr. Alexander Bolotov

September 8, 2015

1 Acknowledgements

I would like to thank my supervisor Dr Alexander Bolotov for constantly helping me out the entire way through the project. I'd also like to thank my brother Hamed Baraky.

List of Figures

List of Tables

Contents

2 Introduction

The aim of this project is to create and provide a foundation for anyone who's interested in using gestures to control a robot. This project being a proof of concept will be designed such that anyone can take segments of this project and manipulated to suit their own needs.

2.1 What Is This Project about?

This project is about, proving the concept that a robot can be controlled by gestures using mainstream technology such as a Microsoft Kinect. It is also about providing a foundation for anyone who wishes to use gesture control to manipulate robotics. All the source code and content of this project will be released online and free to use by anyone. Along with a guide on how to build the robotics and different components used. Ultimately, it hopes to aid understanding and contribute towards the field of gesture controlled robotics.

2.2 Why Choose This Project/Previous Work

I chose this project because I'm quite interested by the field of robotics and I wanted to do something that would give me an idea of what this field was like. I have seen the Microsoft Kinect be used for many things, and wanted to see if it was possible to control a remote robot. In the past I have done some electronic works, such as making small circuitry's and tinkering with hardware, but never anything on this scale. In terms of programming language I've never used Python but i have used C#. Most of this project is going to be new to me.

2.3 Aims and Objectives

The aim of this project is to produce a Windows application will can use the Microsoft Kinect in order to track your movements and respond to certain gestures. Along with the Windows application. There will be a robot, this

robot will respond the gestures detected by the Windows application and respond appropriately. The different gestures will be raising your hands to move forward, the higher you raise your hand the foster the robot will travel. Placing your hands at the centre of your body would move the robot in reverse, how wide you stretch your hands will determine the speed. Raising one hand up and placing the other by your side will cause the robot to turn in that direction, i.e. if you raise your left hand up the robot should turn left.

3 Research

This section is to go through some of the research process that was done during this project. Creating something which is better than what already exists/that which may not even exist starts with a very long researching period. Many different aspects are to be considered some of the more important ones are covered in this chapter.

3.1 System Overview

Based on the proposal, from the very beginning, it is clear that this project will require two main sections in the system. The Robot side of the system, and the remote Computational Engine of the system.

Robot

The robot section of the system consists of two separate parts, the physical robot and the software. Each play a critical role that will rely on each other.

Physical

The physical section of the robot consists of the physical robot itself, that includes all the hardware such as motors, gearbox, infrared sensors, batteries etc. The hardware has to be able to allow the robot to, navigate around under its own power (no human intervention), be able to see and detect objects in front or behind,

provide sufficient power for the onboard computer (Raspberry Pi) and other electronic components.

Software

The software section of the robot consists of, all the programming necessary in order to utilize the hardware which will allow the robot to successfully navigate around a room. T(o avoid colliding in with objects all whilst being instructed remotely by a laptop using a Kinect sensor.

Computational Engine

On the Computational Engine side of our system, we will have a Microsoft Kinect which will be plugged into a laptop/desktop, which is on the same network as the robot. The application will use the Kinect to detect a person and track their skeletal movements. The application will be looking for specific gestures such as raising both arms up, once it has detected a gesture, the application will determine what command this gesture relates to i.e. the direction they wish to robot to navigate and how fast. It will then transmit this command to the robot via the network.

3.2 System Architecture

For this project, it was decided that a component-based architecture would best suited. Robotic systems are often component-based because it helps programmers in terms of coding and scalability. In a component-based system a larger system is built from smaller subcomponents in contrast to an object orientated architecture in which you'd create abstract classes, then inherits from them in order to reuse code. Component-based works particularly well with robotics because the different sections are created in a modular fashion, meaning that you can take a specific section, from one project and use in another because all the different sections can be used independently. The main modules of my project are the Windows application, the FSM(finite

state machine) and the hardware controller.

Windows Application

The Windows application uses a Kinect device to track gestures and control the robot but it can also be used to control any robotic device, so long as the robot is designed to accept numbers as commands. The application comes with an installer and can be used on any Windows machine. The application does not depend on the user specifically having this robot, the commands outputted are generic and can be used to do other things. An example of a command that this application will output is 01-50, in the case of this project, these numbers are translated inside the robot to mean go forward at speed 50.

FSM

The FSM is used to transition from one state to another, i.e. from Idle to Move, it is what translates the command's sent over the network from the Windows application, the command sent from the application are in number form , i.e. 01–50, which translates into next state it should transition into. The FSM is separated from the hardware controller, meaning you could increase or change the different hardware components without having to change the state machine.

Hardware Controller

The hardware controller is used to initialize and communicate with all low-level hardware, all the different pieces of code for the hardware are collected into methods which can then be used by other parts of a system. The hardware controller has been separated from the FSM in order to make it more modular, i.e. if the number of wheels on the robot were to be changed from 4 to 6 all that would be required would be to add the new motors into the movement method, leaving the rest of the system, unaffected.

3.3 Methodology

The software lifecycle/methodology is used to guide the different phases of the development as well as what sequence those phases are arranged in. There are about five main phases that every lifecycle goes through, Requirements and Analysis, Design, Implementation, Testing and deployment. The selection of software lifecycle has a big impact on how the software will be developed, not choosing a correct software lifecycle can lead to, unnecessary delays, unhappy clients, incompleted projects , etc.

The most traditional lifecycle method Is the Waterfall, which consists of a linear sequence of phases mentioned above. Other alternatives are models such as V-Model, Incremental and agile. After analysing the different phases used in these methods, it was deemed that none of them was very suitable. The lifecycle that was chosen for this project was Prototyping.

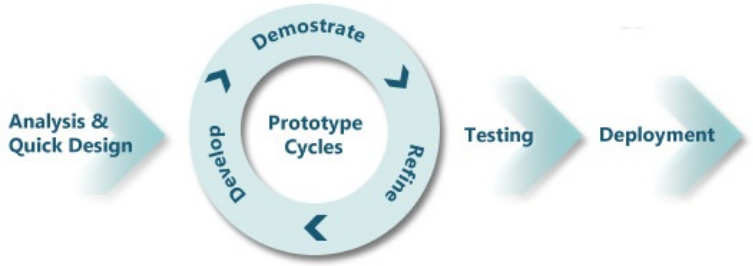

Figure 1: Prototyping Lifecycle

As shown in the figure, prototyping consists of a initial analysis phase, followed by a cycle of prototyping where you are constantly developing and improving until the desired result is reached, then comes the testing and deployment phase.

One of the reasons why prototyping was chosen as opposed to the others, is because hardware can be temperamental and difficult to setup, especially if there is no prior knowledge in that area. And as such significant changes were expected to be constantly made to the original requirements. The prototyping

methodology allows for continuous trial of hardware to see what works and what will have to be changed. In contrast to other methodologies such as Waterfall where requirements are not often changed after they are sent. Ultimately there is a higher degree of tolerance when it comes to making changes and alterations with Prototyping, which is why it was chosen.

3.4 Programming Language

The programming language to be used was carefully considered as it plays a key role in how the software will be developed. Due to the nature of the project being robotic and the uncertainty that comes putting together different low-level electronic devices and making them work together, the main question that was asked during the language selection process was 'Will this language support what it is I wanted to?' and ' how much support is there?' As the project is split into two main sections. I also split the language selection.

Laptop The hardware that's going to be used on the laptop side of the project is the Microsoft Kinect, a device which contains a infrared scanner, colour sensor and depth Sensor. Microsoft provides an SDK for the Kinect in three different languages C++, C# and Visual Basic. The second part of the laptop side consists of a Windows application, this narrows down the choice to C# and Visual Basic because although you can produce a Windows application with C++, it would require a lot more time than using the other two languages. Ultimately the choice was to use C# because of the support it has in terms of libraries for the Kinect , as well as its support for networking, which is what we will be using to send the command's to the robot.

Robot The onboard computer for the robot is going to be a Raspberry Pi, all the motors and sensors will be plugged into it and so all of our program to control these different components it will be done through the Raspberry Pi. Many different languages have support for low-level devices plugged

into the Pi e.g. C, very basic C#, Ruby, Perl, Python ect. The three main competitors were Java, C and Python. The list was then narrowed down to Java and Python. In the end Python was chosen due to support it has in terms of libraries for hardware control, as opposed to Java.

3.5 Hardware Selection

Choosing the right bits of hardware for the robot is critical in order to make it function properly and as desired. Often the challenge with robotic-based projects is the sheer volume of different components that could be used and figuring out whether these components are compatible with one another. Much research was done in this aspect, but it was still within reason to expect several parts not to work well with one another and that they would have to be replaced or changed.

Gesture control: In terms of gesture control, It was decided that a Microsoft Kinect will be used in the project proposal. The reason why a Kinect was chosen is because it's mainstream and widely available, anyone who would want to re-create this project would find it easy to get access to one as most people who have an Xbox usually also have a Kinect. The Kinect is already widely used within the Xbox to track a user's movements and gestures for games and other applications. Inside the Kinect there is an array of microphones, which can be used for voice recognition tasks, there is an IR sensor, a colour sensor and a depth sensor. These sensors combined can be used by software to gather a 3-D depth image, which can then be processed in many different ways.

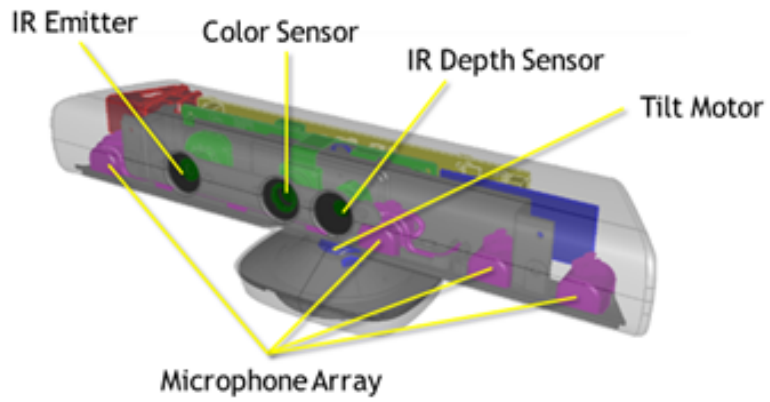

Figure 2: Microsoft Kinect exposed

Currently on the market there are three different types of the Kinect, Kinect for Xbox, Kinect for Windows and Kinect for Xbox one. Kinect for the Xbox one was excluded from the options because being relatively new, not a lot of people would have access to, there is less support available and the device is quite expensive. The Kinect for Xbox and Windows are almost the same with the exception of the Windows version having a feature called "Near Mode" which allows the sensor to work at closer ranges. For this project the Kinect for Windows was used because it was available to use from the University.

Movement and Obstacle Detection: Looking at the project proposal. We can see that it is expected of the robot to be able to move through a room and not collide with obstacles. The movement aspects can be solved by the use of motors and wheels. When it comes to small robotic vehicles. There are three main types of motors, DC motors, Servo motors and Stepper motors. Servo motors are usually used for small mounted devices that require rotation, e.g., an ultrasonic scanner, which spins 180 degrees left and right would be placed on a Servo motor. Servo motors have a limited range of movement and so cannot be used to spin the wheels of the robot. Stepper motors are very similar to Servo motors, the difference being how the internal mechanisms

spin the axle, Stepper motors have a continuous very precise rotation - making them more accurate than Servo and DC motors, at the cost of speed. Servo motors and stepper motors, are usually more expensive than DC motors, DC motors are continuous rotation motors the speed of which is controlled using PWM(pulse width modulation), they usually work at a higher rpm(rotations per minute) , making them a good choice for driving the wheels of a robot car.

In terms of obstacle detection. There are three different types of sensors, laser sensors, ultrasonic sensors and infrared sensors. Laser and ultrasonic sensors are often used in robotics to measure longer distances. Ultrasonic sensors release pulses of sounds and then waits for the sound pulses to return, using the time and the speed of sound you can calculate the distance the pulse traveled. Laser sensors use a similar technique, but instead of pulses of sounds, they released bursts of light. Infrared sensors emit a pulse of infrared light than wait to see if the light is reflected off the surface and comes back. Infrared sensors are good at detecting proximity, the disadvantaged being that they can only tell you an obstacle is within a certain range - which is usually quite limited about 30cm in the case of the IR sensor used in this project. For this project infrared sensors were chosen, because they are cheap and provide close range detection.

Onboard Processor: Choosing the correct onboard computer and motor shield for this project is important because it's going to be driving all the motors and reading in values from the sensors. For a robotic project, there are several main types of onboard computers, micro-processes, microcontrollers and SBC's(single-board computer). The microcontroller that was originally used in the first versions of the prototyping was a Arduino Uno. It's a small microcontroller that cost around £15. Although it is fully capable of controlling all the hardware, it had limitations in terms of processing power and RAM. The advantage of using the Arduino Uno is that because there is no operating system, all the code is written directly onto the board, it's very good for real-time applications and it has excellent support for GPIO but adding

network capability to the Arduino was challenging, extra parts was required, such as a Wi-Fi shield , which is quite expensive and not within the budget.

In the end it was decided to use a SBC called Raspberry Pi instead. The Raspberry Pi has 1Gb of RAM, a quad core processor, four USB ports and 40 GPIO pins for this project. Unlike the Arduino, the raspberry pie runs an operating system typically Linux. Connecting the raspberry pie to the network is much simpler than the Arduino, using the USB ports provided a simple, cheap Wi-Fi dongle allows for network connectivity. The raspberry pie also supports video output via HDMI. But for the majority of the time of the project VNC was used to remotely work on the raspberry pie.

As well as the Raspberry Pi, a piece of circuitry called a motor shield is required. This is because the GPIO pins on the Raspberry Pi can only output a maximum of 50 mA across all the pins. This is fine for low-power hardware like the IR sensors. But this will not be enough power to drive the motors, and so a separate piece of circuitry is required to provide extra power to the motors and other sensors. The motor shield that was chosen is a PiRoCon 2.0. It fits directly on top of the Raspberry Pi and can be used to power the Pi as well as itself.

4 Requirements

This chapter of the report covers the requirements process in this project.

4.1 Robot Requirements

These are the different requirements required by the robot

4.1.1 Functional Requirements

FR1. The robot shall be self-propelling. The robot shall be about to move by its self and without the need of a person pushing it or pulling it.

FR2. The robot shall be able to detect a collision. The robot shall

be able to utilize hardware(IR sensor) in order to detect a collision and stop before the collision occurs.

FR3. The robot shall be able to move around. The robot shall be able to physically move around with the use of motors and wheels.

FR4. The robot shall be able to connect to a wireless network. The robot shall be able to use a Wi-Fi dongle in order to connect to a wireless network

FR5. The robot shall be able to accept connections from a remote laptop. The robot shall be setup with SSH enabled, so that remote devices may connect

FR6. The robot shall be able to translate and execute sent command's. The robot shall be able to read the command sent by the Windows application and translate this to the appropriate reaction i.e. moving forward.

4.1.2 Non-functional Requirements

NFR1. The robot shall have a dedicated power source and charging unit. The robot shall use an array of batteries to power itself, and implement a charger to easily replenish the batteries.

NFR2. The robot shall incorporate a finite state machine architecture. The robot shall use a FSM style architecture to transist from its different states.

4.1.3 Use Case

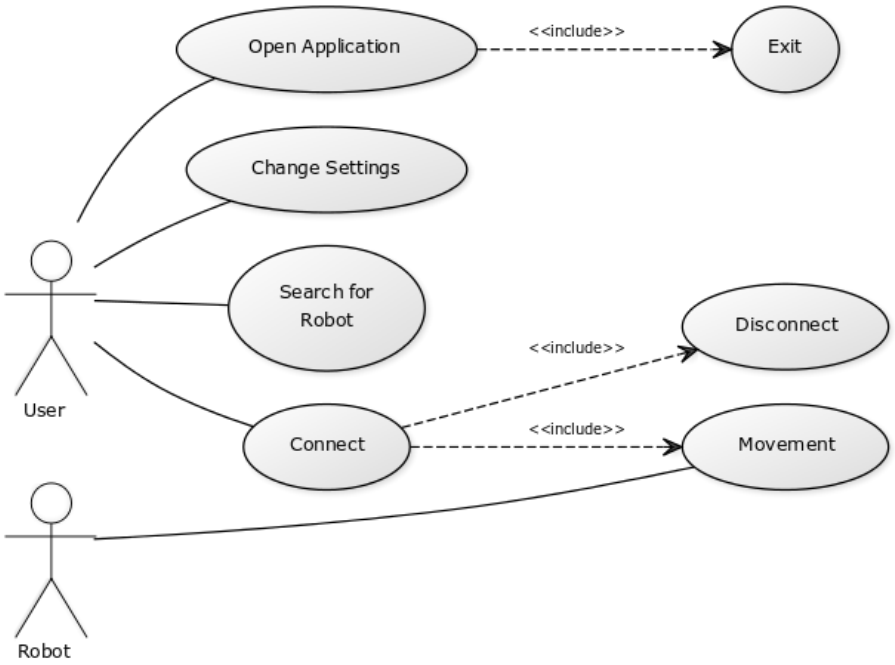

Figure 3: Laptop Use-Case

4.2 Computational Engine Requirements

These are different requirements required by the Windows application.

4.2.1 Functional Requirements

FR1. The application shall allow the user to search for the robot on the network. The application shall allow the user to automatically scan network and find the robot, removing the need to find the IP address of the robot and manually connecting to it.

FR2. The application shall allow the user to change advanced settings such as hostname, username and password. The application shall easily allow the user to change the host name that the application searches for as well as the username and password so it may match the username and

password of the robot.

FR3. The application shall allow the user to connect to the robot via the network. The application shall allow A user to connect to the robot that's on the network , which will begin the gesture control feature.

FR4. The application shall provide the user with manual control of the robot. The application shall have a dedicated section to manual control, which will allow the user to control the robot without the Kinect/gestures control.

FR5.The application shall make use of the Microsoft Kinect. The application shall detect and be able to use the Microsoft Kinect.

FR6.The application shall display the video output of the Kinect. The application shall display the colour video outputs so that the user can tell if the Kinect is working and what it can see.

FR7.The application shall display on top of the video output the skeletal outline of the user it is currently tracking. The application shall display the skeletal outline it is tracking so the user can see what the Kinect is detecting.

4.2.2 Non-functional Requirements

NFR1. The application shall be intuitive to use.. The application shall be simple and straightforward to improve the user experience.

NFR2. The application shall be able to run on any Windows system above Windows 7.

5 Design

5.1 Hardware Design

5.1.1 Acquiring

Figure 4: Stock chassis

The first step of the hardware design was to have some sort of chassis that would contain all of the electronics. Originally a chassis was to be constructed from wood, but due to time constrictions it was decided it would be better to buy a premade chassis. After much research, a chassis was found which had the motors that were required- built into the chassis so it was purchased(shown above).

Next was to acquire the sensors that would allow us to detect obstacles, from the same website two IR sensors were purchased. These two IR sensors will be placed on the front and the rear of the vehicle, allowing the robot to detect a collision while travelling forwards or in reverse. The Raspberry Pi and the PiRoCon shield, but also purchased from that website.

Figure 5: Batteries acquired from a old laptop

The batteries that would eventually be used in the final design of the robot, were acquired by taking apart an old

laptop battery, inside of which were six 18650 batteries(shown above).

5.1.2 Power Management

During the design process one of the big objective that was set was how the robot was going to be powered, for the robot to be independent batteries were a must. All the different components of the robot will be either plugged into the Raspberry Pi or the PiRoCon shield (the PCB that plugs into the Pi).

The PiRoCon can power the Raspberry Pi as well as itself, so the best option would be to power the shield and provided there is enough power, it will power all the other devices. In terms of power requirement, the Raspberry Pi requires a constant 5V and depending on how many USB ports and GPIO pins are being used, can require up to 3000 mA of current. The PiRoCon can accept a range from 6v to 9v, it has a built-in regulator which will provide a safe 5 V for the Pi. During the first prototyping phases six AA batteries connected in series was used to power the robot. Each battery was rated at 1.2V 1000mAh, providing a total of 7.2V. Although the voltage was in range it was quickly evident that this battery solution would not be adequate because the device was running out power too quickly. It also required the batteries to be constantly replaced because there was no charging solution. A solution with more capacity was needed.

The idea of using laptop batteries came around whilst talking to a colleague about laptop battery life and how it's improved over the years. Access to several old laptops was available and so by dismantling a battery pack revealed six laptop batteries(18650) which were rated at 3.7V 2600mAh, more than twice the capacity and voltage compared to the AA battery. Because these batteries are rated at 3.7 V putting them in series would result in a total voltage of 22.2V , which is over the limit of what the PiRoCon can handle, anything above 9V risks damage to the electronic components, 22 V would certainly cause serious damage. And so to idea came forth to create two separate clusters, each cluster would contain three batteries connected in parallel

- meaning there voltages would remain constant, but the current would add at together.

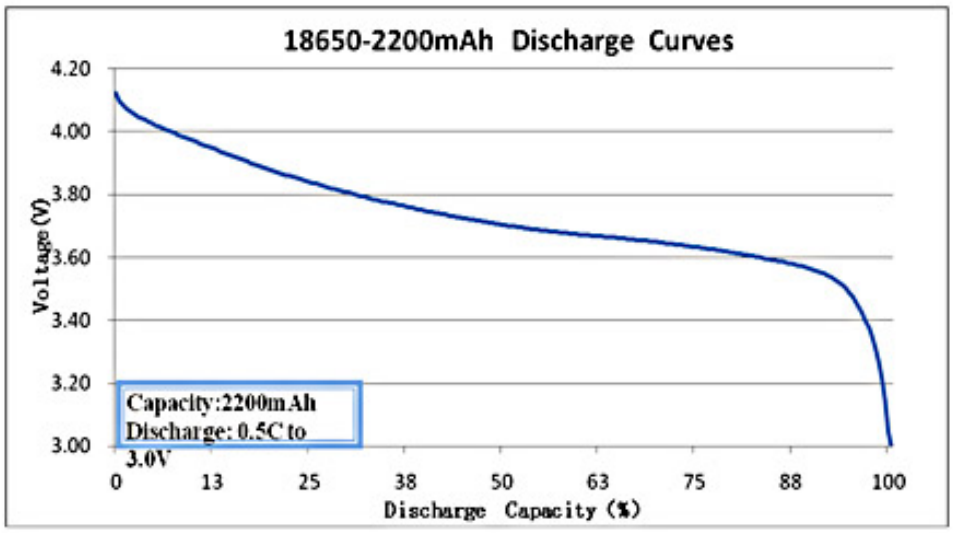

Figure 6: Discharge Curve of a 18650 battery

And then the two clusters will be connected in series - adding the voltage together. On all batteries, the voltage printed on the labels is what's known as a nominal voltage(the average voltage during that batteries charge and discharge cycle), the actual voltage varies depending on what percentage of discharge, the batteries on. This can be represented in the form of discharge curve, as shown on the above. This information is also present on the datasheet that comes with batteries. The batteries acquired from the laptop had a very specific serial code, which led to the finding of the datasheet online (Link in the appendix). From here we can see that voltage of the batteries when at full capacity is 4.2 V, 0.6V higher than the voltage printed on the label. Using this information, the maximum voltage that our proposed battery design will output is 8.4V, which is within the acceptable range. Connected to the battery will be a small voltmeter which will help us when debugging power issues and give us a reading of the voltage. By using the discharge curve and the datasheet . We can figure out at what battery level the voltage will be i.e.

	Battery Packs		
Charge Level	1	2	3
0%	2.8	5.6	8.4
25%	3.7	7.4	11.1
50%	3.8	7.6	11.4
75%	3.9	7.8	11.7
90%	4.0	8.0	12.0
Min Voltage	2.8	5.6	8.4
Max Voltage	4.2	8.4	12.6

Table 1: Charge Level Table

when the battery is at 25%. what should the voltmeter readout.

5.1.3 Charger Design

Now that the battery pack had been designed, it was time to design the how these batteries will be charged, this of course means creating a charger. A simple Google search reveals that there are many different Circuits is out there for a Li-ion charger, it was decided to replicate a very simple and universal design which was found but with slight modification. It began by designing the circuit in, Proteus - which is an application that allows you to design electronic circuitry.

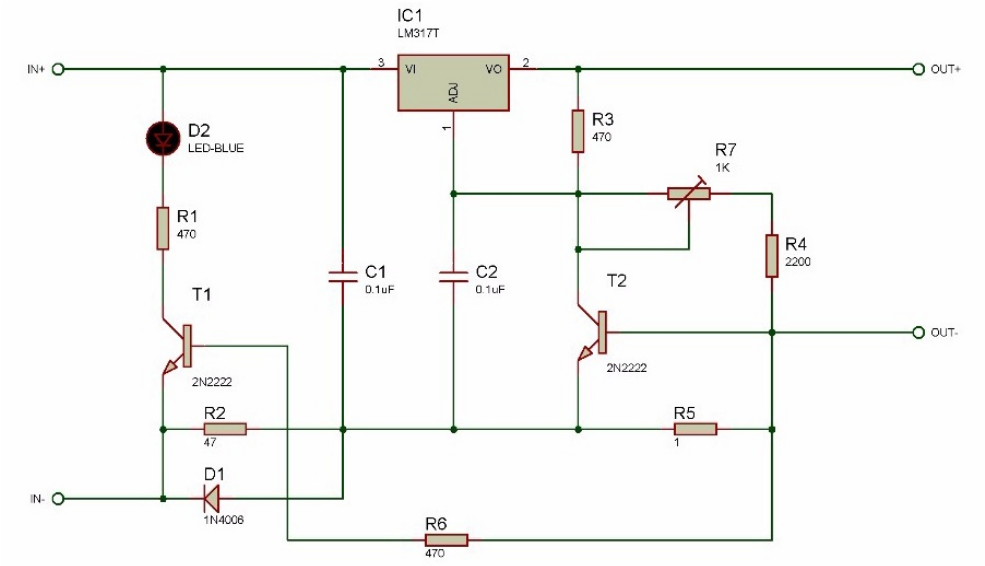

Figure 7: Design schematic of the charger

What this circuit basically does is, accpect 12V DC and reduces it down to 8.4 V, which is the maximum voltage of the battery pack. The circuitry was initially put together on a breadboard to test and make sure that it worked, It was then transferred it over to a through hole PCB. Unfortunately, once transferred over to the PCP the circuit stopped working. Eventually it was discovered that the circuitry was too compacted(shown below).

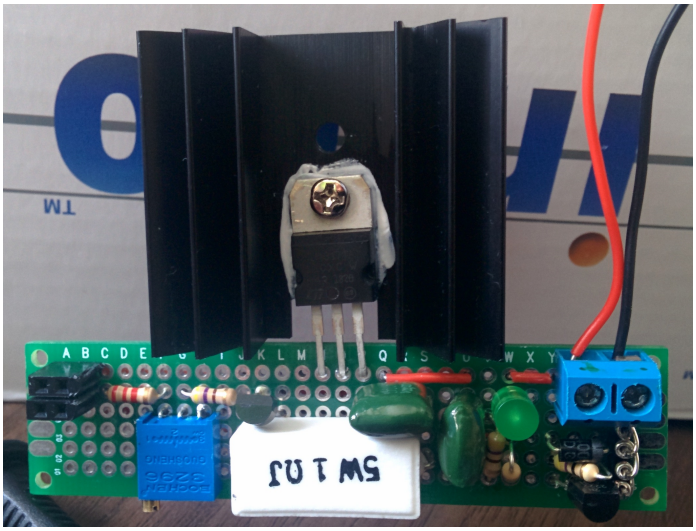

Figure 8: First Attempt that failed

It was constructed again on a larger PCP and it worked(shown below). An additional four slot rails added to the top right of the section of the PCB. These extra slots will be used to power things such as the mini voltmeter.

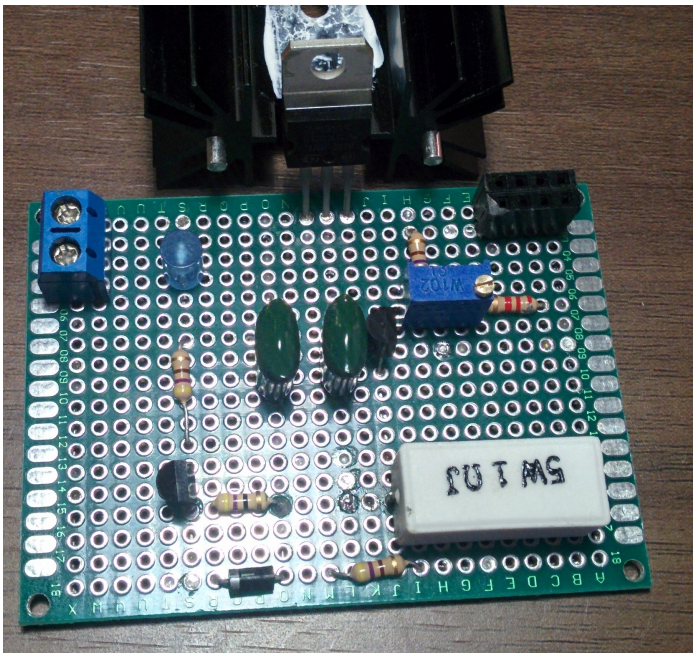

Figure 9: Second Attempt

5.1.4 Layout

The robot consists of two levels and so the layout process began by modelling how the second level was going to be laid out, as seen below the infrared sensors were positioned on the right-hand side of the level facing the front and rear of the car.

In the centre of the level there are two battery packs facing each other. Each battery pack contains three 3.7V 18650 batteries (Laptop batteries) in parallel Both battery packs are in series. At full capacity the total nominal voltage of the packs are 7.4V Which can output a maximum of 45A

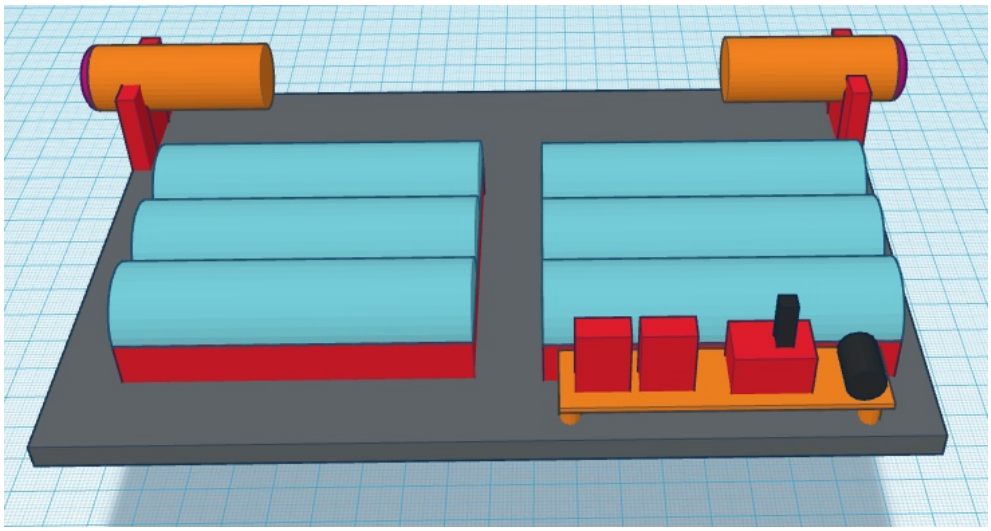

Figure 10: 3D model of Level 2

On level 1, at the front of the robot there will be a small voltmeter marked V which will help us to debug power issues and tell us how much charge the batteries have left. Behind it will be the charging circuitry marked C, which will be used to not only charge the batteries but to provide power to things such as the voltmeter. At the rear of the motor we have the PiRoCon shield mounted on top of the Raspberry Pi marked R.P.

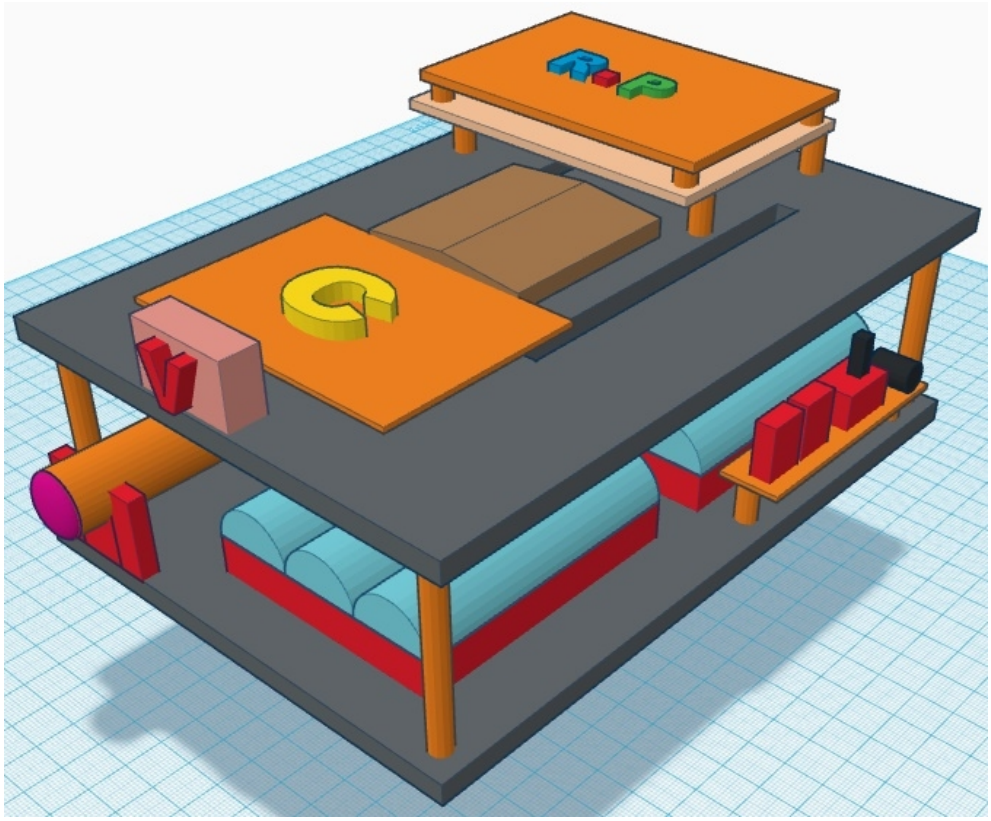

Figure 11: 3D model of Level 1

5.2 Software Design

5.2.1 Network Communication

In order for command's to be sent remotely to the Raspberry Pi, it has to be able to connect to a network. On a hardware level this is done by the use of a Wi-Fi dongle. The first time the Raspberry Pi needs to be connected to a network, it has to be done manually, by plugging in a video display and a mouse and keyboard and then connecting to a Wi-Fi service. Once connected thou, every time the Raspberry Pi boots up, it will attempt to connect to the network if available.

For this project, SSH will be used to communicate and send command's from the Windows application to the robot. SSH is a secure way to access and execute commands on a remote system. The way the networked command

system works is, when the Windows application first connects to the robot an a SSH connection is made. When the application is ready to send a command, it uses the connection to write a certain sequence of numbers within a specific file on the Pi, so for example if the command was move forward at speed 50, the command would be '01-50'. This sequence of numbers will be written inside a specific text file on the Raspberry Pi. A script on the Raspberry Pi(which is first activated when the connection is made) is constantly checking this file and translating this sequence of numbers into transitions.

5.2.2 FSM

The software design process began with designing the coding for the robot. During the research phase, it was decided that a FSM architecture would work best with this kind of robot. In essence, a FSM is any system that stores different states and uses inputs to transist from one state to another. A simple analysis can be carried out to determine what states we will require, looking at the project proposal we can see that the robot has to be able to move so Move would be one of our states. The robot will not always be moving, at some point it's going to remain idle, this gives us the state Idle. At some point the robot will be turned on and off this gives us the states Start and Exit. Also the robot has to be able to detect collisions giving the final state Collision. So the five states are:

- Start
- Idle
- Move
- Collision
- Exit

Now that all the states have been discovered, it must now be considered what input would cause one state to transist to another. In the project the

Windows application will be sending commands over the network to the robot. These commands come in number form(01-05) and have to be translated into transitions. By going through every state and asking what input would cause the state to change one can slowly build up a table. For example, if the system is in state Start , the input that would cause this to change state would be 'toIdle'. From this we can create a state transition table.

	Current State				
Input	Start	Idle	Move	Collision	Exit
toIdle	Idle	Idle	Idle	Idle	—
toMove	—	Move	Move	Move	—
toCollision	—	—	Collision	Collision	—
toExit	—	Exit	Exit	Exit	—

Table 2: Robot's State transition table

A state transition table clearly illustrates all the different transitions, we will have in our system and when they will occur. For instance we can see in the table above in order to change state from Idle to Move, and input of toMove needs to be sent. Using the state table we can also create a state diagram, which can help better visualize the system(shown below)

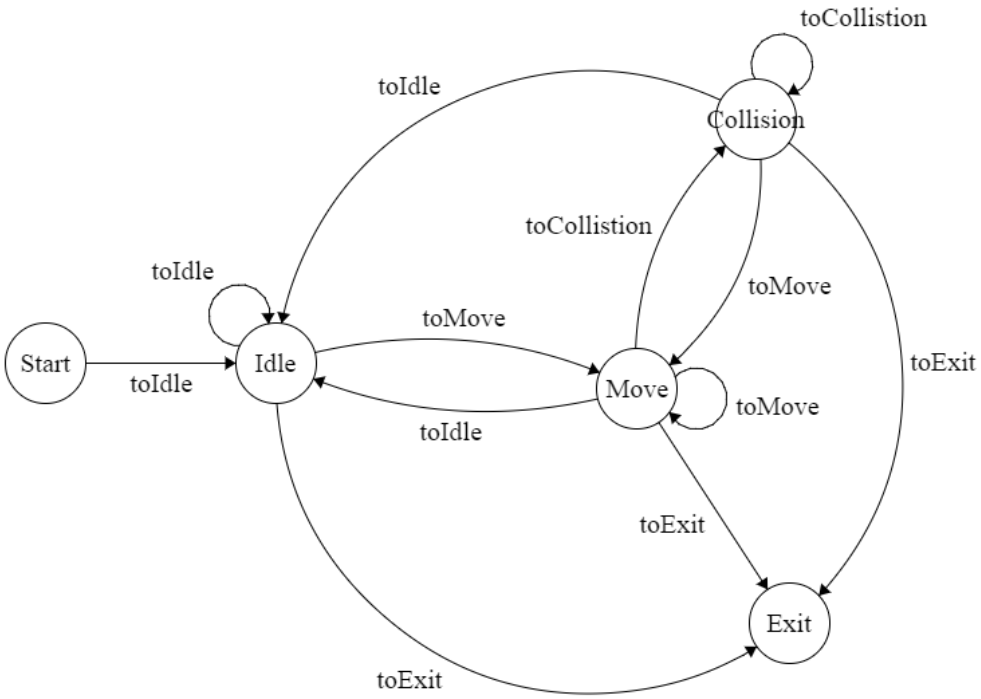

Figure 12: Robot's State Diagram

The way the FSM will be implemented is such that every state there will be three sections of code that will be executed. Code that can be executed when the system first transistors into that state, the main code that is executed and the code that is executed upon exiting a state. Creating a superclass called State from which all the other states would inherit from seemed like the best approach because it will reduce the need to repeat code.

The superclass State also plays another important role - to obtain the commands sent by the Windows application and translate them into transitions. It does this by opening up the file that the Windows application will be communicating to, then reading in the sequence of numbers and split them appropriately. The command sequence comes in two parts. The first set of number indicates what the next transition state should be, and the second number indicates the speed which the robot will travel at.

All the data to do with the states and transitions will be stored in a data class. The data class contains methods which allow the adding of states and

transitions. During the runtime code of script all the states and transitions to be used are initialized. The data class is also responsible for checking to see if there is a new transition and transitioning to it.

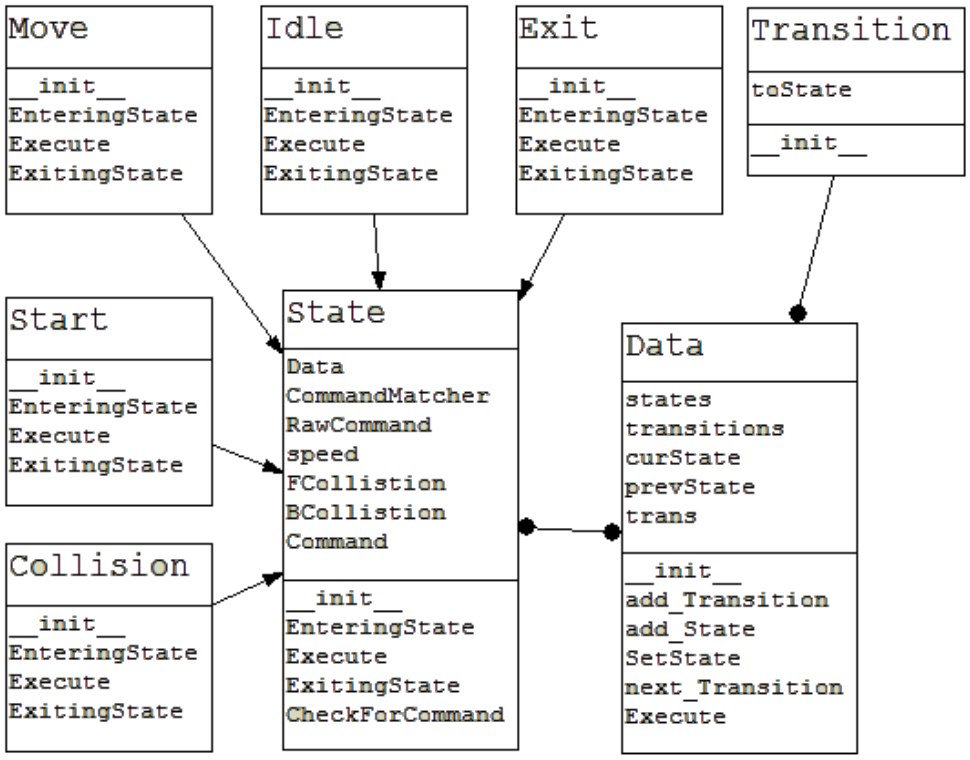

Figure 13: Class Diagram of the FSM

5.2.3 Application

The Windows application, can be split top into three main sections. Establishing and maintaining the connection, using the Microsoft Kinect to gather the necessary data and visually displaying and using skeletal data from the Kinect to recognize gestures.

Connection The initial connection process begins with it searching for the robot on the network. One of the difficulties with working with remote robot is that because IP address changes from network to network, it is required that you manually connect up the robot to a display and find

the IP address, this can be difficult, especially if the robot is inaccessible or the components are enclosed in a chassis. A solution to this problem would be to use the hostname. Most devices that are connected to a network will have a hostname, this is the name assigned by the owner of the device to be able to identify it. Hostnames do not change between network to network unless done so on purpose and so this makes it static. How do host names and IP addresses relate? When a device connects to the network the DNS(Domain Name System) server will add the hostname and the IP addresses to a table. After which, when you use the hostname to connect to something, the request is sent to the modem/server which will resolve the hostname into an IP address by looking through the table.However the disadvantage of using hostname is, networks that are not properly setup can experience hostname resolution conflicts, this is where hostnames don't correctly get resolved to an IP.It can also take time for some DNS server to update their records as not all DNS servers use the same time intervals.

Once the robot has been discovered on the network using the hostname and gets the IP address and the option to connect to the robot is presented to the user. Once the user selects the connect option the IP address along with a username and password will be used to create an SSH connection. The default username and password will be 'pi' and 'ray'. The username and password has to be the same as the one on the robot, if the username or password changes for any reason the user has the option to manually override this and set their own.

For the connection two separate SSH channels will be created in separate threads. The first SSH channel will be running in a separate thread and is used to run the FSM(called CommandReader), this is the script that will be reading the command's and executing them. The second SSH channel which will be running in the main thread will be used to send the command's to the robot. When the application is exited both these

channels will be closed.

Microsoft Kinect The data received from the Microsoft Kinect will be used for two things, firstly to display an output of what the Kinect sees and secondly, using the skeletal data to recognize gestures.Having a visual output helps users to interact with the application better and makes it more exciting. The output that is going to be displayed to the user will have two layers. The first layer will be a standard video output using the colour camera on the Kinect, by showing this output the user receives visual feedback that the device is properly working, it also helps the user to better position himself so that he's in full sight of the Kinect. The second layer will be the skeletal outline that the connect is currently tracking, this layer will be imprinted on top of the first. This allows the user to not only see himself but to see what sections of his body the Kinect is tracking. This will help the user to position his body in a more optimal fashion.

Gesture recognition As soon as a skeletal outline is detected, the method which is responsible for recognizing different gestures is is called, The method will check to see if the robot is connected if it is it will continue, it will then use the skeletal information provided to perform certain comparisons. Not only is the direction of the robot controlled by gestures, but so is the speed. The different gestures that the method recognizes is moving forward this is done by raising both hands above your shoulders, depending on how high you raise your hand the robot will go faster. The second gesture is going in reverse , which is done by placing both hands at the centre of your body the speed is controlled by how wide the hands are extended. The third and fourth gestures are turning left and right. This is done by placing one hand below your hip and raising the other about your shoulder. And the last gesture is stop this is done by placing both hands to your side.

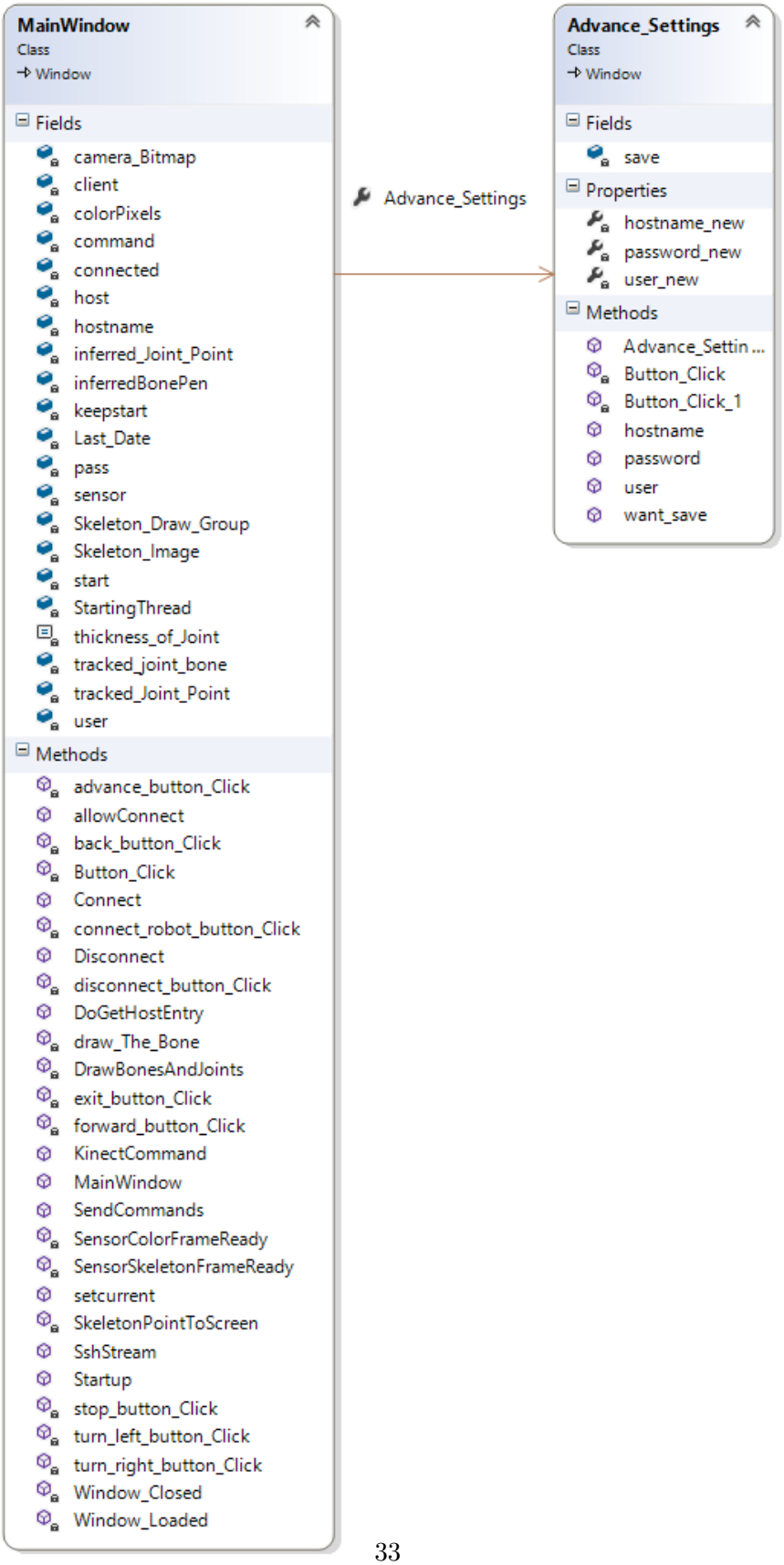

Figure 14: Class Diagram of the Application

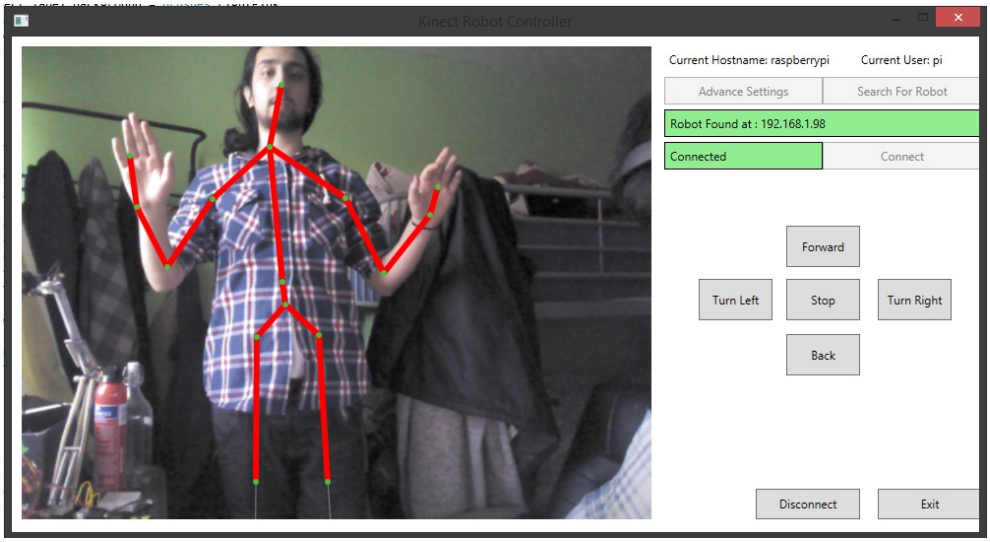

Figure 15: What the final application looks like

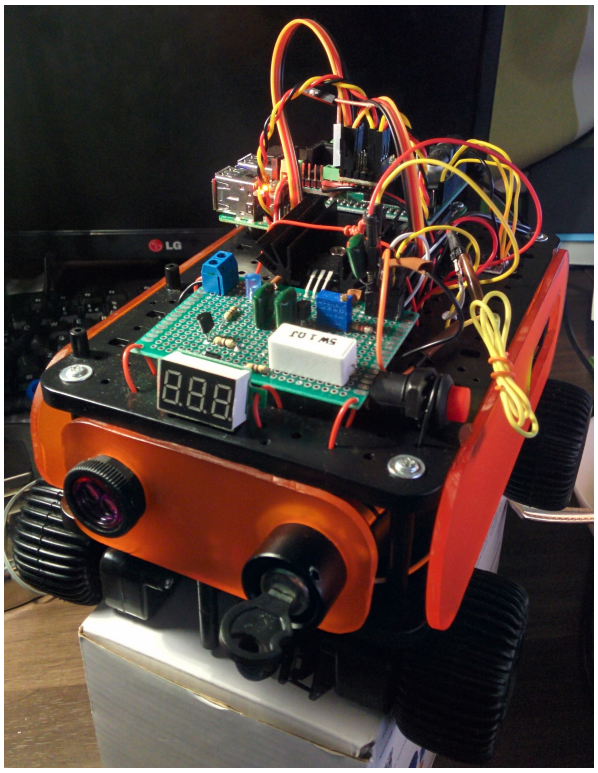

Figure 16: What the final robot looks like

6 Implementation

This section of the report is aimed at providing an insight into some of the more interesting and crucial aspects of the implementation.

6.0.4 Hardware Controller

The hardware controller is the script that's used to directly communicate with the hardware, that includes the infrared sensors on the motors. The implementation began by creating a initialization method, which would initialize all of the senses and motors(as shown in the snippet below)

```
def initialise():
    global ML1, ML2, MR1, MR2
    GPIO.setmode(GPIO.BOARD)

    GPIO.setup(IRBack, GPIO.IN)
    GPIO.setup(IRFront, GPIO.IN)

    GPIO.setup(Left1, GPIO.OUT)
    ML1 = GPIO.PWM(Left1, 20)
    ML1.start(0)
```

This is also where the PWM(pulse width modulation) for the motors was setup. PWM is used to control the motors for a few reasons, first off PWM saves energy, when the robot is powered by battery pack efficiency is a concern, by using PWM we eliminate the need to resist the voltage in order to reduce speed. Using a variable resistor to control speed is inefficient because energy is lost as heat as the resistor limits the voltage is also makes the motors more likely to stall at lower speeds. PWM is much better suited for low-speed vehicles, this is because every pulse is at full power, to the motor has sufficient energy to avoid a stall.In the code above the PWM is set to an initial 20Hz, this may seem rather low. But once the program is running the frequency changes to match the speed the robot needs to travel.

Next a two simple methods were created to return the value of the sensors, the sensor fires short bursts of infrared light and waits to see if the light returns. An infrared sensor is one of the most basic detection sensors. In the code a simple if statement probes the GPIO output of the sensor, give it is equal to zero that means an object has been detected(shown below).

```
def irBack():
    if GPIO.input(IRBack)==0:
        return True
    else:
        return False
```

Next was to setup the methods that would allow us to move the robot. Because we are using PWM, moving the robot is done by adjusting the frequency and duty cycle of the motors. Frequency is the number of cycles per second and the duty cycle is the percentage of how long the signal is active in one cycle, for instance if the frequency was five, and the duty cycle was 25, the motor would only spin for 25% of the cycle, for the rest of the 75% it would be off(Shown right).

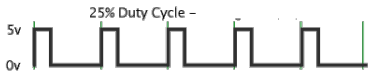

Figure 17: PWM 5Hz 25% Duty Cycle

```
def stop():
    ML1.ChangeDutyCycle(0)
    ML2.ChangeDutyCycle(0)
    MR1.ChangeDutyCycle(0)
    MR2.ChangeDutyCycle(0)
```

In order for the robot to stop I simply set all the motors to have a duty cycle of zero. What this effectively does is provide 0v to the motors.

```
def forward(speed):
    ML1.ChangeDutyCycle(speed)
```

```
    ML2.ChangeDutyCycle(0)
    MR1.ChangeDutyCycle(speed)
    MR2.ChangeDutyCycle(0)
    ML1.ChangeFrequency(speed)
    MR1.ChangeFrequency(speed)
```

In order to move forward the duty cycle and frequency of ML1 and MR2 are set to the speed sent in and ML2 and MR2 is set to a duty cycle of 0. Each motor has two pinouts, so for example the left motor would be assigned ML1(MotorLeft1) and ML2(MotorLeft2) . These two pinouts represent the positive and negative wires going to the motor. And so by activating ML1 and MR1 and disabling ML2 and MR2 . We are creating a positive flow, making the motors spin, forward.

```
def reverse(speed):
    ML1.ChangeDutyCycle(0)
    ML2.ChangeDutyCycle(speed)
    MR1.ChangeDutyCycle(0)
    MR2.ChangeDutyCycle(speed)
    ML2.ChangeFrequency(speed)
    MR2.ChangeFrequency(speed)
```

In order to go reverse the opposite is done, ML2 and MR2 are activated and ML1 and MR1 are set to 0. This switch is the polarity around in the vehicle goes in reverse. Turning left and right is done by, making the right-hand side motors go forward and the left hand side go backwards and vice versa for the other way round.

6.0.5 FSM

Implementing an effective finite state machine was one of the more challenging aspects of this project. It was challenging because it can be quite difficult to visualize such a different type of architecture then what one's used to. The process started by creating the superclass that all the other states would inherit

from, I called this class State, in the constructor. I define a dictionary which have a certain number associated to a transition, e.g. ”00” : ”toIdle”, this dictionary will be used to compare against the command's sent by the Windows application. After which three methods will be defined EnteringState, Execute and ExitingState. These methods are empty, but would normally be used to execute code at different stages of a state.

Then comes the method CheckForCommand, this method is very important because it is in here that the command's from the Windows application is read in and used to change states. Every state will at one point executed this method so they can check what the next command is. Inside the method. First, we open up the file, assign its content to a variable, split that content where hyphen is into two separate numbers. The first number 'Val[0]' is used to determine what the next transition will be, the second number is used to set the speed of the robot. The second number obviously has no effect if the transition is to go into idle or collision. After the numbers have been split to boolens are created and set as false. These boolens will be used to check for collision.

The first number is then passed through the dictionary mentioned before, and the corresponding transition is saved into a variable called Command, after this is done two checks are made . If the command was telling the robot to go forward or backwards. The infrared sensors are checked to see if there's a collision. If there is the Commant variable will be set to 'toCollision ' otherwise, the script will continue. The State Class is shown below.

```
class State(object):
 def __init__(self,Data):
  self.Data = Data
  self.CommandMatcher = {
  "00" : "toIdle",
  "01" : "toMove",
  "02" : "toMove",
  "03" : "toMove",
  "04" : "toMove",
```

```
 "05" : "toExit",
}
def EnteringState(self):
 pass
def Execute (self):
 pass
def ExitingState(self):
 pass
def CheckForCommand(self):
 txt = open("control/Commands.txt")
 value = txt.read()
 val = value.split('-', 1)
 self.RawCommand = val[0]
 self.speed = int(val[1])
 self.FCollistion = False
 self.BCollistion = False
 self.Command = self.CommandMatcher[self.RawCommand]

 if self.RawCommand == "01":
  if hardware_controller.irFront():
   self.Command = "toCollision"
 if self.RawCommand == "02":
  if hardware_controller.irBack():
   self.Command = "toCollision"
```

```
def Execute(self):
 print("In Move")
 super(Move , self).CheckForCommand()
 print(self.Command)

 if self.RawCommand == "01":
  hardware_controller.forward(self.speed)
 if self.RawCommand == "02":
  hardware_controller.reverse(self.speed)
 if self.RawCommand == "03":
  hardware_controller.turnLeft(self.speed)
 if self.RawCommand == "04":
```

```
    hardware_controller.turnRight(self.speed)

  self.Data.next_Transition(self.Command)
```

Shown above, is the method Execute from the state Move, it shows how within the move state the direction's for the robot's movement is determined. It does this by comparing the value of the variable RawCommand, RawCommand is the first set of numbers that comes from the file read in above(from the Windows application). If this value is 01 than the robot will move forward, if the value is set to 02 , the robot will move backwards, if the value is set to 03, the robot will turn left and finally, if the value is set to 04 the robot will turn right. 'self.speed' is the speed at which this robot should this action, and it is set in the superstate, when the command's from the Windows application is read.

```
class Data(object):
 def __init__(self):
  #self.char = character
  self.states = {}
  self.transitions = {}
  self.curState = None
  self.prevState = None
  self.trans = None

 def add_Transition(self,transName, transition):
  self.transitions[transName] = transition

 def add_State(self,stateName,state):
  self.states[stateName] = state

 def SetState(self, stateName):
  self.prevState = self.curState
  self.curState = self.states[stateName]

 def next_Transition(self, toTrans):
```

```
  self.trans = self.transitions[toTrans]

 def Execute(self):
  if(self.trans):
   self.curState.ExitingState()
   self.SetState(self.trans.toState)
   self.curState.EnteringState()
   self.trans = None
  self.curState.Execute()

Data.add_State("Start",Start(Data))
Data.add_State("Idle",Idle(Data))

Data.add_Transition("toIdle", Transition("Idle"))
```

This class called data is what's used to store all of our transitions and states. At the bottom of the snippet above two examples of adding states and one example of adding a transition is given. To begin with, in the constructor there are five things defined.

- A dictionary called states, this will be used to store all the states that will be involved in our system.
- A dictionary called transitions, this will be used to store all the different transitions that can occur in the system.
- A variable called 'curState' which will store the current state that the system is in.
- A variable called 'prevState' which will store the previous state. The system was in.
- A variable called 'trans' which will store the name of the next transition the system will be moving to.

And finally

Next we have four methods, which are effectively setters for the variables we defined in the constructor.

- The first method adds a transition to the dictionary.
- The second method adds a state to the dictionary.
- The third method sets the current state to the previous state variable and then sets the current state to the new state.
- The fourth method sets the variable trans to whatever the next transition will be.
- The final method in this class is called Execute

. This method first checks to see if a new transition has been issued, if it hasn't, it will execute the code of the current state. So for example, if the current state of the system is Idle and no new transitions have been set, when the method Execute is called, it will execute code the code thats in the Idle state. However if a new transition has been issued it will run the exiting code of Current state, it will then set the state of whatever the transition leads to e.g. if the transition is 'toIdle' then it will set the state to Idle. It will then run the EnteringState code, set trans to 'None' (this resets the if condition) , and finally executes the main code of that state.

```
Data.SetState("Start")
while True:
 time.sleep(0.3)
 Data.Execute()
```

Above is is a segment of the code that runs in the main method. What that code does every 0.3 seconds it runs Data.Execute(), what this effectively means is every 0.3 seconds this script is reading in the command, checking what transition it needs to change to and doing that transition and executing the function of the new state.

6.0.6 Robot Connection

In order to establish a connection between the laptop and the robot the system will be using SSH, the programming language C# doesn't inherently have any libraries for SSH and so the use of an external library called SSH.NET was needed.

To begin the process we first define a set of global variables(shown below).

```
string hostname = "raspberrypi";
string host;
string user = "pi";
string pass = "ray";
string command;
SshClient client;
SshClient start;
Boolean connected = false;
Boolean keepstart = false;
Thread StartingThread;
```

- Hostname is what the system will be using to resolve the IP address of the robot.
- Host is where the system will store the IP address.
- User and pass are the username and password to the Raspberry Pi (this can be changed later by the user if he or she wishes).
- Command is the instruction the system will be sending.
- Client and start are shh channels .
- Connected is a boolean used to check whether a connection is made.
- keepstart is a boolean used to make sure FSM on the robot stays active.
- StartingThread is a thread which will contain the first SSH channel, which activates the FSM

When the application first makes a connection to the robot. It starts a new thread called startup and sets the Boolean of keepstart to true. This new thread then creates an SSH connection, and then runs the command which starts the FSM. In theory, the only time this thread should end is when the application ends because the command that activates that FSM should not progress forward. But during several tests , it was found that that every now and again the FSM would close and the thread would exit. After this discovery the Boolean keepstart was added so that if the FSM should unexpectedly close and the connection is still active it reconnects and runs the command.

```
        public void Startup()
        {
            while (keepstart)
            {
                start = new SshClient(host, user, pass);
                try
                {
                    start.Connect();
                    start.RunCommand("sudo python3 control/
                        CommandReader.py");
                }
                catch
                {
                    Console.Write("not wokring");
                }
            } start.Disconnect();
```

The snippet below shows how once a command has been set, how its sent. By the time this section of the code is executed another SSH channel called client has been created, the channel client is what's used to send the new command's. Also the Command that needs to be sent has been assigned to the variable command.

```
public void SshStream()
        {
```

```
            try
            {
                var Command = client.CreateCommand(command);
                var asynch = Command.BeginExecute();
                var output = new StreamReader(Command.
                    OutputStream);
                var error_message = new StreamReader(Command
                    .ExtendedOutputStream);

                while (!asynch.IsCompleted)
                {
                    var result = output.ReadToEnd();
                    var error = error_message.ReadToEnd();
                    if (string.IsNullOrEmpty(result) &&
                        string.IsNullOrEmpty(error))
                        continue;
                    Console.WriteLine("StdErr: " + error);
                    Console.WriteLine("StdOut: " + result);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
```

The first line sets to a variable a client type which has a command that is ready to be executed. Note , it's only setting the variable not sending the command. The second line begins the execution process and assigns the output to a variable called asynch. It is here where the command is being executed , i.e., the command ("echo '05-00' — cat ¿ control/Commands.txt"). The variables output and error_message, get whatever output/error that command gives, i.e. if the command was "ls -l" the output would be a list of all the files and folders in that directory. While the execution isn't finished it's going to be constantly writing the ouput of the command to the variables result and error.

The reason why it was chosen to execute the command this way and not just a simple client.runcommand(""); is because this method provides an output making debugging much easier, during the creation of this application seeing what output, the command was providing was essential to making sure it worked properly. Below shows shows an example of how the system would send the command that instructs the robot to turn left.

```
case 3:
        command = "echo '03-100' | cat > control/Commands.
            txt";
        SshStream();
        break;
```

6.0.7 Kinect Skeleton Overlay

In order to do the skeleton overlay a handful of global variables are declared, including a variable of type KinectSensor, all the different brushes that will be used to draw the skeleton - different coloured brushes will be used for tracked bones/joints and inferred bones/joins. The difference between Inferred and Tracked is, tracked is when the Kinect can actively see the joints and bones, Inferred is when It guesses where a joint/bone will be E.G. If the Kinect can see your wrist and shoulder but not the rest of the arm it will try to guess where it is and draw it. Also declared, is a drawing group and the drawing image. The drawing group will allow us to bundle all the different drawings into one group and drawing image will allow us to use a drawing in a image container.

```
KinectSensor sensor;
const double thickness_of_Joint = 4;
Brush tracked_Joint_Point = Brushes.Green;
Brush inferred_Joint_Point = Brushes.Yellow;
Pen tracked_joint_bone = new Pen(Brushes.Red, 5);
Pen inferredBonePen = new Pen(Brushes.Gray, 2);
DrawingGroup Skeleton_Draw_Group;
```

```
DrawingImage Skeleton_Image;
```

In the Window_Loaded method, the group for the drawing and the drawing image is setup. Then a check is made to see if a Kinect is connected, if so, we enable the skeleton stream and then make an event handler, which is fired every time a new frame is available from the skeleton stream.

```
this.Skeleton_Draw_Group = new DrawingGroup();
this.Skeleton_Image = new DrawingImage(this.
    Skeleton_Draw_Group);
Image_skeleton.Source = this.Skeleton_Image;

this.sensor.SkeletonStream.Enable();//This turns on the
    skeleton stream.
this.sensor.SkeletonFrameReady += this.
    SensorSkeletonFrameReady;
```

When the skeleton event handler is activated, it creates a new skeleton object from the class skeleton, this will be used to contain all the data about that skeleton frame. Next the data about the skeleton, which is currently in the frame is put into this object. After this is done the drawing process begins, a rectangle is drawn on top of the container 'Image' which is in the main window. It is inside this rectangle that all the drawings will be rendered, this rectangle has been purposefully made transparent, so that any skeleton has drawn will be on top of the video output of the Kinect. It is at this point that the method which is used to interpret gestures is called KinectCommand(more about this method later). And lastly a method which will state what sections of the skeleton are to be drawn out.

```
Skeleton[] skeletons = new Skeleton[0];
skeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];
skeletonFrame.CopySkeletonDataTo(skeletons);
dc.DrawRectangle(Brushes.Transparent, null, new Rect(0.0,
    0.0, 640.0f, 480.0f));
```

```
KinectCommand(skel);
this.DrawBonesAndJoints(skel, dc);
```

The actual drawing of the skeleton occurs in a method called draw_the_Bone, in the constructor of this method amongst other things,there are two joints. It is between these two joints that the bone is going to be drawn, for example, the right elbow and the right wrist. The joints that are passed through can been in two states either tracked or inferred. If both the joints sent through are inferred then that it is not drawn, if one of them is tracked. Then an line is drawn between them, and this new drawing is added to the group.

```
(Stripped-down version of the class)
private void draw_The_Bone(Skeleton skeleton,
    DrawingContext drawingContext, JointType jointType0,
    JointType jointType1)
        {
            Joint joint0 = skeleton.Joints[jointType0];
            Joint joint1 = skeleton.Joints[jointType1];

            // If both the joints that are sent in our
                inferred, then we do not draw this
            if (joint0.TrackingState == JointTrackingState.
                Inferred &&
                joint1.TrackingState == JointTrackingState.
                    Inferred)
            {
                return;
            }
            Pen drawPen = this.inferredBonePen;
            if (joint0.TrackingState == JointTrackingState.
                Tracked && joint1.TrackingState ==
                JointTrackingState.Tracked)
            {
                drawPen = this.tracked_joint_bone;
            }
```

```
            drawingContext.DrawLine(drawPen, this.
                SkeletonPointToScreen(joint0.Position), this
                .SkeletonPointToScreen(joint1.Position));
        }
```

6.0.8 Gesture recognition

The gesture recognition is done by a method called KinectCommand, every time a new skeleton frame is detected this method is called. In the constructor it requires that a model of the skeleton be sent. The method then checks to see if it is connected to the robot. It then creates a timing limiter. The aim of this is to prevent gestures command's being sent too frequently to the robot. The current time is stored in a variable and compared against the last time the system was in that method, if that time is less than half a second, then the method exits . Otherwise, the method will continue(Shown below).

```
if (connected) //Whilst the application is connected to the
    robot
            {
                var Current_Date = DateTime.Now;
                if (Current_Date.Subtract(Last_Date).
                    TotalMilliseconds < 500) //We set up a
                    timer so we are not constantly sending
                    command's, a gap of half a second is in
                    place
                    return;
                Last_Date = DateTime.Now;
```

Next we create five variables which are of type Joint, each one of these variables will be equal to its corresponding model. The five variables are:

- Right_hand
- Left_hand
- Right_shoulder

- Left_shoulder
- Hip_center

The reason why we only model these joints is because we will be using the distances between them to determine what gestures being performed, the model of the lower body is not required.

```
Joint Right_hand = model.Joints[JointType.HandRight];
Joint Left_hand = model.Joints[JointType.HandLeft];
Joint Right_shoulder = model.Joints[JointType.ShoulderRight
    ];
Joint Left_shoulder = model.Joints[JointType.ShoulderLeft];
Joint Hip_center = model.Joints[JointType.HipCenter];
```

Now we setup the algorithm that will recognize the different gestures and perform its corresponding function. How it works can be demonstrated easily with some pseudocode(Shown below, code in appendix)

```
// if (The Right_Hand Position is ABOVE The Right shoulder
    AND The left_hand Position is ABOVE the Left shoulder)
// {
//  distance1 = to the Distance between the right hand AND
    shoulder * 100
//  distance2 = to the Distance between the left hand AND
    shoulder * 100
//  speed= distance1 + distance2
//  if(speed is > 100)
//     {
//     speed = 100
//     }
//  SendCommand ("Go Farward", "speed")
// }
//
// if(The Right_hand Position is BELOW the Right shoulder
    AND the Right Hand is ABOVE the hip
//    AND if the Left_Hand Postion is BELOW the Left
    shoulder AND the Left_hand is ABOVE the hip)
```

```
// {
// Speed = to the Distance between the Right hand AND left hand * 100
//   if(speed is > 100)
//     {
//     speed = 100
//     }
// SendCommand ("Go Back", "Speed")
// }
//
// if (Left_Hand Postion is ABOVE The Left shoulder AND Right_Hand is BELOW hip)
// {
//     SendCommand("Turn Left" , 100)
// }
//
// if (Right_Hand Postion is ABOVE The Right shoulder AND Left_Hand is BELOW hip)
// {
//     SendCommand("Turn Right" , 100)
// }
//
// if (Right_Hand Postion is BELOW hip AND Left_Hand Postion is BELOW hip)
// {
//     SendCommand("Stop" , 0)
// }
```

The first if statement is going to check for the gesture which makes the robot go forward. It checks to see whether your right hand is above your right shoulder as well as your left hand being above your left shoulder. If this is true, it then gathers the information that will set the speed of the robot - this depends on how high you raise your hands. First it calculates the distance between your right hand in your right shoulder then the distance between your left hand and left shoulder. It multiplies each one of these by 100 - this is because the distance is measured in meters and so we will have very small

values e.g 0.52. These two distances of an added together to give the speed, another if statement checks to see if the speed is above 100, if so it will set it to 100(100 is our speed limit). And then finally it sends the command off.

The second if statement checks for the gesture which makes the robot could backwards. Similar to the first one, except this time it checks whether both your hands are at the centre of your body, i.e. in between your shoulders and hips. If this is so, then it determines what speed. The speed for going backwards is determined by how wide you stretch your hands. The distance between them will be calculated and multiplied by 100. And again, same as before. If the speed is greater than 100, it will be set to 100. Finally, the command to go backwards is sent.

The third if statement checks for the gesture which makes the robot turns left. It checks to see whether your left hand is above your left shoulder and that your right hand facing down, i.e., below your hip. If this is so, then the command to turn left is sent with a speed of 100. Initially the turn speed could be varied depending on how high you raise the arm, but after prototyping it was discovered that that anything below the speed of 90 the motors would not have enough power to spin the robot. And so it was decided to manually set the speed of the turns to 100. The fourth if statement is the same as the third, except it checks to gesture for turning right. Which is the opposite of the third if statement, i.e. it checks to see if the right hand is raised instead of the left.

And finally, the last if statement checks to see if both your hands are below your hip. If this is so it will send the command to stop.

7 Testing

During this project there was an extensive amount of testing done, especially on the robotic side of the project. Sometimes hardware can be very temperamental, especially when integrating several systems together. Every time something was changed or altered slightly the system would be tested because

when it comes to robotics small changes can have a very big impact on the overall system. Another point which was constantly tested was the integration of the Windows application and the robot. Integration can be very difficult between two systems specially over a network and as such, also very temperamental.

A lot of the testing that had to be conducted was physical testing. Things such as, is the robot moving correctly, is the speed being correctly configured, does the robot when it confronts an obstacle. The other side is testing was mostly visual-based i.e., is to connect outputting the correct video, is the skeleton overlaid been correctly positioned. The final and most tricky side of the testing was, testing and network connectivity, i.e. if the hostname was being resolved correctly - for a long while it wouldn't work, even though it was seemed to be correct.

7.0.9 Requirements Met?

7.0.10 Robot Requirements

ID	Functional Requirement	Achieved	Tested
FR1	The robot shall be self-propelling	Yes	Passed
FR2	The robot shall be able to detect a collision	Yes	Passed
FR3	The robot shall be able to move around	Yes	Passed
FR4	The robot shall be able to connect to a wireless network	Yes	Passed
FR5	The robot shall be able to accept connections from a remote laptop	Yes	Passed
FR6	The robot shall be able to translate and execute sent command's	Yes	Passed

Table 3: Met Requirements

ID	Non-Requirement	Achieved	Tested
NFR1	The robot shall have a dedicated power source and charging unit	Yes	Passed
NFR2	The robot shall incorporate a finite state machine architecture	Yes	Passed

Table 4: Met Requirements

7.0.11 Computational Engine Requirements

ID	Functional Requirement	Achieved	Tested
FR1	The application shall allow the user to search for the robot on the network	Yes	Passed
FR2	The application shall allow the user to change advanced settings such as hostname, username and password	Yes	Passed
FR3	The application shall allow the user to connect to the robot via the network	Yes	Passed
FR4	The application shall provide the user with manual control of the robot	Yes	Passed
FR5	The application shall make use of the Microsoft Kinect	Yes	Passed
FR6	The application shall display the video output of the Kinect	Yes	Passed
FR7	The application shall display on top of the video output the skeletal outline of the user it is currently tracking	Yes	Passed

Table 5: Met Requirements

ID	Non-Requirement	Achieved	Tested
NFR1	The application shall be intuitive to use	Yes	Passed
NFR2	The application shall be able to run on any Windows system above Windows 7	Yes/No	Not Tested

Table 6: Met Requirements

8 Critical evaluation

This section of the report will be going over some of the achievements and criticisms experienced by the author, to effectively provide a summary of what went well, what didn't go well and what could've been done differently, As well as some of the author's personal opinions.

8.0.12 Research

During the initial research stage of the project, I was overwhelmed by the sheer number of different components and electronics I could've been used in my project, I had little expertise and ended up going down the wrong path for a while until I found my way back. Had I consulted, perhaps with a member of staff from the electrical engineering department the initial research phase could've gone smoother. I think one of the good things about this phase was once I start going down the right track. I managed to secure a pretty good foundation and a solid workplan, especially in terms of the architecture with things like the finite state machine. Had I have had more time I would've liked to go on down the route of the omnidirectional robot, which seemed quite interesting as a glance.

8.0.13 Requirements

When originally doing the requirements I had no idea whether some of the requirements I was stating was even possible, but using the experience I had obtained from other modules, in the end I ended up with what seemed like a set of achievable requirements.

8.0.14 Design

One of the more challenging aspects of the design process, was figuring out a way to design all the subcomponents, so that they could be integrated together. This proved to be a lot more challenging than I had anticipated. There were a lot of issues and many of the systems didn't want to play well with each other.

Another challenging aspects was creating the power circuitry, I have had some experience with using and creating small electrical circuits. But the power charging circuit for this robot proved to be quite the challenge, my first couple of attempts failed before creating something that worked. If I would've had more time I would like to have spent a little bit more time working on designing a better power circuit.

8.0.15 Implementation

The implementation of this project was hard to say the least, on the surface the concepts look very simple but actually getting down to it, and doing is as something else. I found that most of my time was spent testing the system, every change or code added required the system to be tested due to its temperamental state.

One of the big achievements during this project was achieved during the implementation stage and that was simply getting everything to integrate with one another correctly andand managing to fulfill all my requirements.

8.0.16 Testing

The testing phase for this project was pretty much conducted all the way through, which helped to identify and resolve many issues fairly quickly. Even so, some more time with allow me to even out some of the remaining kinks.

9 Conclusion

In conclusion, I believe this prototype device was very successful, it lays a good foundation for anyone who wishes to replicate this project, to take care and modify it into anything they want. The current build I've made with this idea has been on a small scale, but could be extended up to larger systems. The process of being able to take something from just an idea into a fully finished final product has been incredible. I have learned so much during this process,

especially because a lot of this was new to me and I'm thankful to have had the opportunity.

10 References

() Software Prototyping, Available at: http://en.wikipedia.org/wiki/Software_prototyping (Accessed: 27042015).

() What are the software development models, Available at: http://istqbexamcertification.com/what-are-the-software-development-models/ (Accessed: 27/04/2015).

() Raspberry Pi low level peripherals, Available at: http://elinux.org/RPi_Low-level_peripherals#GPIO_Code_examples (Accessed: 27/04/2015).

() Microsoft Kinect libraries, Available at: https://msdn.microsoft.com/en-us/library/hh973074.aspx (Accessed: 27/04/2015).

() Li-ion Discharge curves, Available at: http://blogberlinmd.com/photorii/li-ion-discharge-curve (Accessed: 27/04/2015).

() Pulse width modulation, Available at: http://www.arduino.cc/en/Tutorial/PWM (Accessed: 27/04/2015).

() Raspberry Pi power requirements, Available at: https://www.raspberrypi.org/forums/viewtopic.php?f=31&t=18564 (Accessed: 27/04/2015).

() Microsoft Kinect SDKs, Available at: http://blogs.msdn.com/b/kinectforwindows/archive/2012/12/07/inside-the-newest-kinect-for-windows-sdk-infrared-control.aspx (Accessed: 27/04/2015).

11 Appendix

11.1 hardware controller

```
import RPi.GPIO as GPIO, threading, os, time, sys,
    subprocess


#Setting the pins for the motors. A single motor as two pin
    outs
#allowing you to change the polarity which lets you change
    the direction
#it spins. The PiRoCon shield i'm using is 24,26 for left
    motor and
#19,21 for the right.
Right1 = 24
Right2 = 26

Left1 = 19
Left2 = 21

#Setting the forward and rear infrared sensors
IRBack = 7
IRFront = 11

def initialise():
 #created some global var's for the motors, ML1 = Motor Left
      1
    global ML1, ML2, MR1, MR2
 #Setting how the IO pins will be numbered. GPIO.BOARD is
    the
 #numbers printed on the board. An alternate setmode is GPIO
    .BCM
 #which uses the channel numbers.
    GPIO.setmode(GPIO.BOARD)

 #Setting up the IR inputs so we can read value of the GPIO
    pin
```

```
    GPIO.setup(IRBack, GPIO.IN)
    GPIO.setup(IRFront, GPIO.IN)

 #Setting up the motors and then enabling pwm
    GPIO.setup(Left1, GPIO.OUT)
    ML1 = GPIO.PWM(Left1, 20)
    ML1.start(0)

    GPIO.setup(Left2, GPIO.OUT)
    ML2 = GPIO.PWM(Left2, 20)
    ML2.start(0)

    GPIO.setup(Right1, GPIO.OUT)
    MR1 = GPIO.PWM(Right1, 20)
    MR1.start(0)

    GPIO.setup(Right2, GPIO.OUT)
    MR2 = GPIO.PWM(Right2, 20)
    MR2.start(0)

def cleanup():
    stop()
    GPIO.cleanup()

def irBack():
    if GPIO.input(IRBack)==0:
        return True
    else:
        return False

def irFront():
    if GPIO.input(IRFront)==0:
        return True
    else:
        return False
#
    ######################################################################
```

```
def stop():
    ML1.ChangeDutyCycle(0)
    ML2.ChangeDutyCycle(0)
    MR1.ChangeDutyCycle(0)
    MR2.ChangeDutyCycle(0)

def forward(speed):
    ML1.ChangeDutyCycle(speed)
    ML2.ChangeDutyCycle(0)
    MR1.ChangeDutyCycle(speed)
    MR2.ChangeDutyCycle(0)
    ML1.ChangeFrequency(speed)
    MR1.ChangeFrequency(speed)

def reverse(speed):
    ML1.ChangeDutyCycle(0)
    ML2.ChangeDutyCycle(speed)
    MR1.ChangeDutyCycle(0)
    MR2.ChangeDutyCycle(speed)
    ML1.ChangeFrequency(speed)
    MR1.ChangeFrequency(speed)

def turnLeft(speed):
    ML1.ChangeDutyCycle(0)
    ML2.ChangeDutyCycle(speed)
    MR1.ChangeDutyCycle(speed)
    MR2.ChangeDutyCycle(0)
    ML2.ChangeFrequency(speed)
    MR1.ChangeFrequency(speed)

def turnRight(speed):
    ML1.ChangeDutyCycle(speed)
    ML2.ChangeDutyCycle(0)
    MR1.ChangeDutyCycle(0)
    MR2.ChangeDutyCycle(speed)
    ML1.ChangeFrequency(speed)
```

```
    MR2.ChangeFrequency(speed)
```

11.2 FSM

```
import time, os, sys, hardware_controller

hardware_controller.initialise()

#This class is used to determine what state we will be
#transitioning to next. The Constructor takes in a string
#containing the name of the next state E.G Idle or move
class Transition(object):
 def __init__(self,toState):
  self.toState= toState
#
    ######################################################################

##States
#This section contains all the different states we will have
#and the code they execute when where in that state.

#This class "State" is a generic superclass from which all
    the other
#states will inherit


class State(object):
 def __init__(self,Data):
  self.Data = Data
  self.CommandMatcher = {
  "00" : "toIdle",
  "01" : "toMove",
  "02" : "toMove",
  "03" : "toMove",
  "04" : "toMove",
  "05" : "toExit",
 }
```

```
 def EnteringState(self):
  pass

 def Execute (self):
  pass

 def ExitingState(self):
  pass

 def CheckForCommand(self):
  txt = open("control/Commands.txt")
  value = txt.read()
  val = value.split('-', 1)
  self.RawCommand = val[0]
  self.speed = int(val[1])
  self.FCollistion = False
  self.BCollistion = False
  self.Command = self.CommandMatcher[self.RawCommand]

  if self.RawCommand == "01":
   if hardware_controller.irFront():
    self.Command = "toCollision"
  if self.RawCommand == "02":
   if hardware_controller.irBack():
    self.Command = "toCollision"



class Start(State):
 def __init__(self, Data):
  super(Start, self).__init__(Data)

 def EnteringState(self):
  pass

 def Execute(self):
  print("In start")
  with open('state.txt','w') as f:
```

```
    f.write("Started")
   time.sleep(1)
   if os.path.isfile('control/Commands.txt'):
    pass
   else:
    open('control/Commands.txt',"w").close()
   self.Data.next_Transition("toIdle")

 def ExitingState(self):
  with open('state.txt','w') as f:
   f.write("Loading")

class Idle(State):
 def __init__(self, Data):
  super(Idle, self).__init__(Data)

 def EnteringState(self):
  with open('state.txt','w') as f:
   f.write("Idle")

 def Execute(self):
  print("In Idle")
  super(Idle , self).CheckForCommand()
  hardware_controller.stop()
  if self.Command == "toIdle":
   pass
  else:
   self.Data.next_Transition(self.Command)

 def ExitingState(self):
  pass

class Move(State):
 def __init__(self, Data):
  super(Move, self).__init__(Data)

 def EnteringState(self):
```

```
105     with open('state.txt','w') as f:
         f.write("Moving")

    def Execute(self):
     print("In Move")
110  super(Move , self).CheckForCommand()
     print(self.Command)

     if self.RawCommand == "01":
      hardware_controller.forward(self.speed)
115  if self.RawCommand == "02":
      hardware_controller.reverse(self.speed)
     if self.RawCommand == "03":
      hardware_controller.turnLeft(self.speed)
     if self.RawCommand == "04":
120   hardware_controller.turnRight(self.speed)

     self.Data.next_Transition(self.Command)

    def ExitingState(self):
125  pass

class Collision(State):
    def __init__(self, Data):
     super(Collision, self).__init__(Data)
130

    def EnteringState(self):
     with open('state.txt','w') as f:
      f.write("Path Blocked")

135 def Execute(self):
     super(Collision , self).CheckForCommand()
     hardware_controller.stop()

     if self.RawCommand == "01":
140   if not hardware_controller.irFront():
       hardware_controller.forward(self.speed)
```

```
    if self.RawCommand == "02":
     if not hardware_controller.irBack():
      hardware_controller.reverse(self.speed)
    if self.RawCommand == "03":
     hardware_controller.turnLeft(self.speed)
    if self.RawCommand == "04":
     hardware_controller.turnRight(self.speed)

    #print(self.Command)
    self.Data.next_Transition(self.Command)

  def ExitingState(self):
   pass

class Exit(State):
 def __init__(self, Data):
  super(Exit, self).__init__(Data)

 def EnteringState(self):
  print("In Exit")
  with open('state.txt','w') as f:
   f.write("Exit")

 def Execute(self):
  hardware_controller.cleanup()
  sys.exit(0)

 def ExitingState(self):
  pass


#
    ##########################################################################

##Data
#This section contains the class in which all the data will
    be kept
```

```
#such as all the transitions, states, current state ect.
#this class also has the def's which allow you to add
    Transitions
#and states
class Data(object):
 def __init__(self):
  #self.char = character
  self.states = {}
  self.transitions = {}
  self.curState = None
  self.prevState = None
  self.trans = None

 def add_Transition(self,transName, transition):
  self.transitions[transName] = transition

 def add_State(self,stateName,state):
  self.states[stateName] = state

 def SetState(self, stateName):
  self.prevState = self.curState
  self.curState = self.states[stateName]

 def next_Transition(self, toTrans):
  self.trans = self.transitions[toTrans]

 def Execute(self):
  if(self.trans):
   self.curState.ExitingState()
   self.SetState(self.trans.toState)
   self.curState.EnteringState()
   self.trans = None
  self.curState.Execute()
#
    ########################################################################

## Runup code
```

```
if __name__=='__main__':
 Data = Data()
 ##States
 Data.add_State("Start",Start(Data))
 Data.add_State("Idle",Idle(Data))
 Data.add_State("Move",Move(Data))
 Data.add_State("Collision",Collision(Data))
 Data.add_State("Exit",Exit(Data))

 ##Transistion
 Data.add_Transition("toIdle", Transition("Idle"))
 Data.add_Transition("toMove", Transition("Move"))
 Data.add_Transition("toCollision", Transition("Collision"))
 Data.add_Transition("toExit", Transition("Exit"))

 Data.SetState("Start")
 while True:
  time.sleep(0.3)
  Data.Execute()
```

11.3 Windows Application- Main

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.IO;
```

```
using Renci.SshNet;
using System.Net;
using System.Threading;
using System.Diagnostics;

using Microsoft.Kinect;

namespace Kinect_Robot
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        string hostname = "raspberrypi";
        string host;
        string user = "pi";
        string pass = "ray";
        string command;
        SshClient client;
        SshClient start;
        Boolean connected = false;
        Boolean keepstart = false;
        Thread StartingThread;

        DateTime Last_Date;

        KinectSensor sensor;//This initialise a variable of
            type KinectSensor which we will use downbelow
        const double thickness_of_Joint = 4;//This is used
            to set the thickness of the dots reprsenting
            joints when we make the skeleton model
        Brush tracked_Joint_Point = Brushes.Green;//This is
            the colour of the dots reprsenting joints E.G.
            The neck, wrist ect
        Brush inferred_Joint_Point = Brushes.Yellow;//This
            is the colour of the dots reprsenting inferred
```

```
        joints E.G. joints it cant see but guess its
        there.
    Pen tracked_joint_bone = new Pen(Brushes.Red, 5);//
        This is the colour and line thickness of the
        lines that will connect the joints and make the
        bones.
    Pen inferredBonePen = new Pen(Brushes.Gray, 2);//
        This is the colour and line thickness of the
        lines that will connect the joints and make the
        bones which is guessed.
    /// <summary>
    /// Inferred vs Tracked. Tracked is when the Kinect
        can activly sees the joints, Infferred is when
        It guesses where a joint will be E.G. If the
        Kinect can see
    /// your wrist and sholder but not the rest of the
        arm it will try to guess where it is and draw it
        .
    /// </summary>

    DrawingGroup Skeleton_Draw_Group; //This
        DrawingGroup allows me to bundle all the
        drawings into one group
    DrawingImage Skeleton_Image; //A DrawingImage type
        allows use to use a drawing in a image E.G.
        Skeleton_Draw_Group

    WriteableBitmap camera_Bitmap; //This creates a
        bitmap which we can keep writing pixels to, we
        will use this bitmap to display the camera
        output.
    byte[] colorPixels;


    public MainWindow()
    {
        InitializeComponent();
```

```
        }

        public Advance_Settings Advance_Settings
        {
            get
            {
                throw new System.NotImplementedException();
            }
            set
            {
            }
        }


        private void Window_Closed(object sender, EventArgs
            e)
        {
            if (connected)
            {
                Disconnect(); // If there is a connection
                    the robot, disconnect upon exit
            }

            if (null != this.sensor)
            {
                this.sensor.Stop();// If there is a Kinect
                    connected, disconnect upon exit
            }
        }

        private void Window_Loaded(object sender,
            RoutedEventArgs e)
        {
            robot_connect_label.Content = "Not Connected";
            robot_connect_label.Background = Brushes.
                LightPink;
            robot_found_label.Content = "Results";
```

```
            current_host.Content = "Current Hostname: " +
                hostname;
            current_user.Content = "Current User: " + user;
            Last_Date = DateTime.Now;

            this.Skeleton_Draw_Group = new DrawingGroup();//
                This will create a group for our drawing.

            this.Skeleton_Image = new DrawingImage(this.
                Skeleton_Draw_Group);//This sets up the
                DrawingImage to use our drawing
                Skeleton_Image;

            // Display the drawing using our image control
            Image_skeleton.Source = this.Skeleton_Image; //
                This sets our skeleton image, to the image
                on the Windows form

            foreach (var potentialSensor in KinectSensor.
                KinectSensors)
            {
                //At the beginning of the application, the
                    application is going to look for the
                    Kinect sensor. If it finds it it's going
                     to set it to the variable sensor.
                if (potentialSensor.Status == KinectStatus.
                    Connected)
                {
                    this.sensor = potentialSensor;
                    break;
                }
            }

            if (null != this.sensor)
            {
                ////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////

// Turn on the color stream to receive color
    frames
this.sensor.ColorStream.Enable(
    ColorImageFormat.
    RgbResolution640x480Fps30); // This
    enables the colour stream, so we can get
    video output

// Allocate space to put the pixels we'll
    receive
this.colorPixels = new byte[this.sensor.
    ColorStream.FramePixelDataLength];

this.camera_Bitmap = new WriteableBitmap(
    this.sensor.ColorStream.FrameWidth, this
    .sensor.ColorStream.FrameHeight, 96.0,
    96.0, PixelFormats.Bgr32, null);
//This is the bitmap that we are going to
    display on the screen.

this.Image_cam.Source = this.camera_Bitmap;
    //This sets the bitmap to the image on
    the Windows form.


this.sensor.ColorFrameReady += this.
    SensorColorFrameReady;//This is an event
    handler which is activated every time
    there is a new frame

////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
```

```
            // Turn on the skeleton stream to receive
                skeleton frames
            this.sensor.SkeletonStream.Enable();//This
                turns on the skeleton stream.

            this.sensor.SkeletonFrameReady += this.
                SensorSkeletonFrameReady;
            //This is an event handler which is
                activated every time it detects there is
                 a new skeleton in the frame.

            // Start the sensor!
            try
            {
                this.sensor.Start();
            }
            catch (IOException)
            {
                this.sensor = null;
            }
        }

        if (null == this.sensor)
        {

        }
    }
    ////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////

    private void SensorColorFrameReady(object sender,
        ColorImageFrameReadyEventArgs e)
    {
        //This is the event handler, for the video
            output of the connect.
```

```
165             //Every time there is a new frame this method is
                    called, and the new frame is written into
                   the bitmap. Which then appears on the screen
                using (ColorImageFrame colorFrame = e.
                    OpenColorImageFrame())
                {
                    if (colorFrame != null)
                    {
170                     // Copy the pixel data from the image to
                            a temporary array
                        colorFrame.CopyPixelDataTo(this.
                            colorPixels);

                        //Writes the image data into the bitmap
                        this.camera_Bitmap.WritePixels(
175                         new Int32Rect(0, 0, this.
                                camera_Bitmap.PixelWidth, this.
                                camera_Bitmap.PixelHeight),
                            this.colorPixels,
                            this.camera_Bitmap.PixelWidth *
                                sizeof(int),
                            0);
                    }
180             }
            }


            //This is the event handler for the skeleton
                tracking. It is called every time there's a new
                skeleton in the frame.
185         void SensorSkeletonFrameReady(object sender,
                SkeletonFrameReadyEventArgs e)
            {

                Skeleton[] skeletons = new Skeleton[0];
```

```
190             using (SkeletonFrame skeletonFrame = e.
                    OpenSkeletonFrame())
                {
                    if (skeletonFrame != null)
                    {
                        skeletons = new Skeleton[skeletonFrame.
                            SkeletonArrayLength];
195                     skeletonFrame.CopySkeletonDataTo(
                            skeletons);
                    }
                }

                using (DrawingContext dc = this.
                    Skeleton_Draw_Group.Open())
200             {
                    //This section is responsible for drawing
                        the skeleton outline.
                    // We create a transparent rectangle set to
                        the same size as the image on the main
                        form 640x480.
                    //The rectangles is made transparent, so the
                         skeleton overlays the video output
                    dc.DrawRectangle(Brushes.Transparent, null,
                        new Rect(0.0, 0.0, 640.0f, 480.0f));
205
                    if (skeletons.Length != 0)
                    {
                        foreach (Skeleton skel in skeletons)
                        {
210
                            if (skel.TrackingState ==
                                SkeletonTrackingState.Tracked)
                            {
                                KinectCommand(skel);//Every time
                                     there is new skeletal data,
                                     this sends the
```

```
                        //data to a method called
                            KinectCommand , which
                            searches for gestures.
215                     this.DrawBonesAndJoints(skel, dc
                            ); //This method sets what
                            part of the skeleton is to
                            be drawn
                    }
                }
            }

220         // This prevents the drawings from being
                done outside of our specified area 640
                x480
            this.Skeleton_Draw_Group.ClipGeometry = new
                RectangleGeometry(new Rect(0.0, 0.0,
                640.0f, 480.0f));
        }
    }

225

    // This method defines what section of the skeleton
        is going to be drawn, for our purposes, we only
        need to waist up
    void DrawBonesAndJoints(Skeleton skeleton,
        DrawingContext drawingContext)
    {
        // This will draw out, the torso of the body -
            this is required because our hand gestures
            will
230     //be compared against the hip. And so the user
            can see what section of his upper body is
            being tracked
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.Head, JointType.ShoulderCenter);
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.ShoulderCenter, JointType.
```

```
            ShoulderLeft);
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.ShoulderCenter, JointType.
            ShoulderRight);
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.ShoulderCenter, JointType.Spine);
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.Spine, JointType.HipCenter);
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.HipCenter, JointType.HipLeft);
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.HipCenter, JointType.HipRight);

        // This will draw out the left arm
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.ShoulderLeft, JointType.ElbowLeft)
            ;
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.ElbowLeft, JointType.WristLeft);
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.WristLeft, JointType.HandLeft);

        // This will draw out the Right arm
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.ShoulderRight, JointType.
            ElbowRight);
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.ElbowRight, JointType.WristRight);
        this.draw_The_Bone(skeleton, drawingContext,
            JointType.WristRight, JointType.HandRight);

        //The defined skeleton, calls the method
            draws_the_bone, passing through the two
            points,
        //which the bone we drawn from e.g. the left
            shoulder to the left elbow
```

```
            //  This method creates the dots that will
                represent joints on the overlay. It will
                allow the user to see exactly how many joins
                 of being tracked
            foreach (Joint joint in skeleton.Joints)
            {
                Brush drawBrush = null;

                if (joint.TrackingState ==
                    JointTrackingState.Tracked)
                {
                    drawBrush = this.tracked_Joint_Point;//
                        This sets the attract joints
                }
                else if (joint.TrackingState ==
                    JointTrackingState.Inferred)
                {
                    drawBrush = this.inferred_Joint_Point;//
                        And this sets the inferred joints
                }

                if (drawBrush != null)
                {
                    drawingContext.DrawEllipse(drawBrush,
                        null, this.SkeletonPointToScreen(
                        joint.Position), thickness_of_Joint,
                         thickness_of_Joint);
                }
            }
        }


        //Because the skeleton is being mapped on a Live,
            the user may be moving back
        //and forward as well as left and right, this means
            that the distance from the
```

```
        //camera(depth) is changing. This method makes sure
            that our skeleton render is
        //within our output resolution of 640x 480 and is
            not distorted.
        private Point SkeletonPointToScreen(SkeletonPoint
            skelpoint)
        {
            // We are converting the point in which we want
                to display the joint, to take into the count
                 of the depth of the user.
            DepthImagePoint depthPoint = this.sensor.
                CoordinateMapper.
                MapSkeletonPointToDepthPoint(skelpoint,
                DepthImageFormat.Resolution640x480Fps30);
            return new Point(depthPoint.X, depthPoint.Y);
        }


        //This is the method used to draw a line between two
             points, the last two arguments.
        //The constructor accepts- are the points from which
             lines will be drawn e.g., right elbow to right
            shoulder
        private void draw_The_Bone(Skeleton skeleton,
            DrawingContext drawingContext, JointType
            jointType0, JointType jointType1)
        {
            //The argument of JointType can either be
                tracked or inferred.
            Joint joint0 = skeleton.Joints[jointType0];
            Joint joint1 = skeleton.Joints[jointType1];

            // If the joints that are sent cannot be found,
                then the method quits, exit
            if (joint0.TrackingState == JointTrackingState.
                NotTracked ||
```

```
            joint1.TrackingState == JointTrackingState.
                NotTracked)
        {
            return;
        }

        // If both the joints that are sent in our
            inferred, then we do not draw this
        if (joint0.TrackingState == JointTrackingState.
            Inferred &&
            joint1.TrackingState == JointTrackingState.
                Inferred)
        {
            return;
        }

        // Whilst drawing the bones, every bone is
            considered inferred unless both the joints
            are tracked
        Pen drawPen = this.inferredBonePen;
        if (joint0.TrackingState == JointTrackingState.
            Tracked && joint1.TrackingState ==
            JointTrackingState.Tracked)
        {
            drawPen = this.tracked_joint_bone;
        }

        drawingContext.DrawLine(drawPen, this.
            SkeletonPointToScreen(joint0.Position), this
            .SkeletonPointToScreen(joint1.Position));
    }

    //This is the method used to, to see if any gestures
         are being performed.
    public void KinectCommand(Skeleton model)
    {
```

```
            if (connected) //Whilst the application is
                connected to the robot
            {
                var Current_Date = DateTime.Now;
                if (Current_Date.Subtract(Last_Date).
                    TotalMilliseconds < 500) //We set up a
                    timer so we are not constantly sending
                    command's, a gap of half a second is in
                    place
                    return;
                Last_Date = DateTime.Now;

                //if (StartingThread.IsAlive) { Debug.
                    WriteLine("Its alive"); } else { Debug.
                    WriteLine("Its dead"); }

                Joint Right_hand = model.Joints[JointType.
                    HandRight];
                Joint Left_hand = model.Joints[JointType.
                    HandLeft];
                Joint Right_shoulder = model.Joints[
                    JointType.ShoulderRight];
                Joint Left_shoulder = model.Joints[JointType
                    .ShoulderLeft];
                Joint Hip_center = model.Joints[JointType.
                    HipCenter];


                if (Right_hand.Position.Y > Right_shoulder.
                    Position.Y && Left_hand.Position.Y >
                    Left_shoulder.Position.Y)
                {
                    var distance1 = (Right_hand.Position.Y -
                        Right_shoulder.Position.Y) * 100;
                    var distance2 = (Left_hand.Position.Y -
                        Left_shoulder.Position.Y) * 100;
                    var speedF = distance1 + distance2 + 15;
```

```
            int speed = Convert.ToInt32(speedF);
            Debug.WriteLine("Forward" + speed);
            if (speed > 100)
            {
                speed = 100;
            }
            SendCommands("01", speed);
        }

        if (Right_shoulder.Position.Y > Right_hand.
            Position.Y && Right_hand.Position.Y >
            Hip_center.Position.Y &&
            Left_shoulder.Position.Y > Left_hand.
                Position.Y && Left_hand.Position.Y >
                 Hip_center.Position.Y)
        {
            var speedF = ((Right_hand.Position.X -
                Left_hand.Position.X) * 100);
            int speed = Convert.ToInt32(speedF);
            if (speed > 100)
            {
                speed = 100;
            }
            SendCommands("02", speed);
            Debug.WriteLine("Back" + speed);
        }

        if (Left_hand.Position.Y > Left_shoulder.
            Position.Y && Right_hand.Position.Y <
            Hip_center.Position.Y)
        {
            Debug.WriteLine("Turn Left");
            SendCommands("03", 00);
        }
        if (Left_hand.Position.Y < Hip_center.
            Position.Y && Right_hand.Position.Y >
            Right_shoulder.Position.Y)
```

```
370                 {
                        Debug.WriteLine("Turn Right");
                        SendCommands("04", 00);
                    }
                    if (Right_hand.Position.Y < Hip_center.
                        Position.Y && Left_hand.Position.Y <
                        Hip_center.Position.Y)
375                 {
                        Debug.WriteLine("Stop");
                        SendCommands("05", 00);
                    }

380             }
            }

            /// <summary>
            ///
                ////////////////////////////////////////////////////////////////////////

385         ///
                ////////////////////////////////////////////////////////////////////////

            /// </summary>
            /// <param name="sender"></param>
            /// <param name="e"></param>

390         private void Button_Click(object sender,
                RoutedEventArgs e)
            {
                DoGetHostEntry(hostname);

            }
395
            private void connect_robot_button_Click(object
                sender, RoutedEventArgs e)
            {
                StartingThread = new Thread(() => Startup());
```

```
            keepstart = true;
            StartingThread.Start();
            Connect();
        }

        private void advance_button_Click(object sender,
            RoutedEventArgs e)
        {
            Advance_Settings page = new Advance_Settings();
            page.ShowDialog();
            if (page.want_save())
            {
                hostname = page.hostname();
                user = page.user();
                pass = page.password();
                setcurrent();
            }
        }
        private void disconnect_button_Click(object sender,
            RoutedEventArgs e)
        {
            Disconnect();
        }
        private void forward_button_Click(object sender,
            RoutedEventArgs e)
        {
            SendCommands("01", 80);
        }

        private void back_button_Click(object sender,
            RoutedEventArgs e)
        {
            SendCommands("02", 80);
        }

        private void turn_left_button_Click(object sender,
            RoutedEventArgs e)
```

```
        {
            SendCommands("03", 85);
        }

        private void turn_right_button_Click(object sender,
            RoutedEventArgs e)
        {
            SendCommands("04", 85);
        }

        private void stop_button_Click(object sender,
            RoutedEventArgs e)
        {
            SendCommands("05", 00);
        }

        private void exit_button_Click(object sender,
            RoutedEventArgs e)
        {
            if (connected)
            {
                Disconnect();
            }
            Application.Current.Shutdown();
        }
        ////////////////////////////////////////////////////////////////////////

        ////////////////////////////////////////////////////////////////////////

        //Classes for ssh connection

        public void SendCommands(string direction, int speed
            )
        {
            int dir_number = Int32.Parse(direction);
            switch (dir_number)
            {
```

```
                case 1:
                    command = "echo '01-" + speed + "' | cat
                        > control/Commands.txt";
                    SshStream();
                    break;
                case 2:
                    command = "echo '02-" + speed + "' | cat
                        > control/Commands.txt";
                    SshStream();
                    break;
                case 3:
                    command = "echo '03-100' | cat > control
                        /Commands.txt";
                    SshStream();
                    break;
                case 4:
                    command = "echo '04-100' | cat > control
                        /Commands.txt";
                    SshStream();
                    break;
                case 5:
                    command = "echo '00-00' | cat > control/
                        Commands.txt";
                    SshStream();
                    break;
            }
        }

        public void Startup()
        {
            while (keepstart)
            {
                start = new SshClient(host, user, pass);
                try
                {
                    start.Connect();
```

```
                start.RunCommand("sudo python3 control/
                    CommandReader.py");
            }
            catch
            {
                Console.Write("not wokring");
            }
        } start.Disconnect();


    }

    public void Connect()
    {
        robot_connect_label.Content = "Connected";
        client = new SshClient(host, user, pass);
        try
        {
            client.Connect();
            disconnect_button.IsEnabled = true;
            advance_button.IsEnabled = false;
            search_robot_button.IsEnabled = false;
            connect_robot_button.IsEnabled = false;
            back_button.IsEnabled = true;
            forward_button.IsEnabled = true;
            turn_left_button.IsEnabled = true;
            turn_right_button.IsEnabled = true;
            stop_button.IsEnabled = true;
            robot_connect_label.Background = Brushes.
                LightGreen;
            connected = true;
        }
        catch
        {
            connect_robot_button.IsEnabled = false;
            robot_connect_label.Background = Brushes.
                LightPink;
```

```
                robot_connect_label.Content = "Could not
                    connect to Robot";
                robot_found_label.Background = Brushes.
                    LightPink;
                robot_found_label.Content = "Try resetting
                    the robot";
            }
        }

        public void Disconnect()
        {
            keepstart = true;
            connected = false;
            client.RunCommand("echo '05-00' | cat > control/
                Commands.txt");
            client.RunCommand("echo '00-00' | cat > control/
                Commands.txt");
            client.Disconnect();
            start.Disconnect();
            disconnect_button.IsEnabled = false;
            advance_button.IsEnabled = true;
            search_robot_button.IsEnabled = true;
            connect_robot_button.IsEnabled = false;
            back_button.IsEnabled = false;
            forward_button.IsEnabled = false;
            turn_right_button.IsEnabled = false;
            turn_left_button.IsEnabled = false;
            stop_button.IsEnabled = false;
            robot_connect_label.Content = "Not Connected";
            robot_connect_label.Background = Brushes.
                LightPink;
            robot_found_label.Content = "Search again...";
            robot_found_label.Background = Brushes.
                LightYellow;
            StartingThread.Abort();

        }
```

```
        public void DoGetHostEntry(string hostname)
        {
            Thread.Sleep(1000);
            try
            {
                IPHostEntry hostdata = Dns.GetHostEntry(
                    hostname);
                IPAddress ip = hostdata.AddressList[0];
                robot_found_label.Background = Brushes.
                    LightGreen;
                robot_found_label.Content = "Record Found at
                     : " + ip;
                host = ip.ToString();
                allowConnect();

            }
            catch (Exception)
            {
                robot_found_label.Background = Brushes.
                    LightPink;
                robot_found_label.Content = "Robot not Found
                     on Network";
            }
        }

        public void allowConnect()
        {
            connect_robot_button.IsEnabled = true;
        }

        public void setcurrent()
        {
            current_host.Content = "Current Hostname: " +
                hostname;
            current_user.Content = "Current User: " + user;
        }
```

```
        public void SshStream()
        {
            try
            {
                var Command = client.CreateCommand(command);
                var asynch = Command.BeginExecute();
                var output = new StreamReader(Command.
                    OutputStream);
                var error_message = new StreamReader(Command
                    .ExtendedOutputStream);

                while (!asynch.IsCompleted)
                {
                    var result = output.ReadToEnd();
                    var error = error_message.ReadToEnd();
                    if (string.IsNullOrEmpty(result) &&
                        string.IsNullOrEmpty(error))
                        continue;
                    Console.WriteLine("StdErr: " + error);
                    Console.WriteLine("StdOut: " + result);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
        ////////////////////////////////////////////////////////////////////////////////

        ////////////////////////////////////////////////////////////////////////////////

    }
}
```

11.4 Windows Application- Advance Setting

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Kinect_Robot
{
    /// <summary>
    /// Interaction logic for Advance_Settings.xaml
    /// </summary>
    public partial class Advance_Settings : Window
    {
        string hostname_new { get; set; }
        string user_new { get; set; }
        string password_new { get; set; }
        Boolean save = false;
        public Advance_Settings()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender,
            RoutedEventArgs e)
        {
            hostname_new = new_hostname.Text;
            user_new = new_user.Text;
            password_new = new_password.Text;
            save = true;
```

```
            this.Close();
        }

        public string hostname()
        {
            return hostname_new;
        }

        public string user()
        {
            return user_new;
        }

        public string password()
        {
            return password_new;
        }
        public Boolean want_save()
        {
            return save;
        }

        private void Button_Click_1(object sender,
            RoutedEventArgs e)
        {
            this.Close();
        }
    }
}
```